Modeling Advanced Security Aspects of Key Exchange and Secure Channel Protocols

Vom Fachbereich Informatik der
Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades
Doctor rerum naturalium (Dr. rer. nat.)
von

Felix Günther, M.Sc. M.Sc.
geboren in Heppenheim

Referenten: Prof. Dr. Marc Fischlin
Prof. Dr. Kenneth G. Paterson

Tag der Einreichung: 12. Dezember 2017
Tag der mündlichen Prüfung: 6. Februar 2018

Darmstadt, 2018
D 17

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit – abgesehen von den in ihr ausdrücklich genannten Hilfen – selbständig verfasst habe.

Wissenschaftlicher Werdegang

Oktober 2007 – November 2010

Studium der Informatik an der Technischen Universität Darmstadt,
Bachelor of Science, Note: mit Auszeichnung.

November 2010 – März 2013

Studium der Informatik an der Technischen Universität Darmstadt,
Master of Science, Note: mit Auszeichnung.

November 2010 – März 2013

Studium der Informatik - IT Security an der Technischen Universität Darmstadt,
Master of Science, Note: mit Auszeichnung.

seit Juni 2013

Doktorand der Informatik an der Technischen Universität Darmstadt.

List of Publications

Journal Articles

[1] Felix Günther and Bertram Poettering. Linkable Message Tagging: Solving the Key Distribution Problem of Signature Schemes. *International Journal of Information Security (IJIS)*, 16(3):281–297, 2017. Also available as Cryptology ePrint Archive Report 2014/014, `https://eprint.iacr.org/2014/014`.

[2] Jean Paul Degabriele, Victoria Fehr, Marc Fischlin, Tommaso Gagliardoni, Felix Günther, Giorgia Azzurra Marson, Arno Mittelbach, and Kenneth G. Paterson. Unpicking PLAID: a cryptographic analysis of an ISO-standards-track authentication protocol. *International Journal of Information Security (IJIS)*, 15(6):637–657, 2016. Also available as Cryptology ePrint Archive Report 2014/728, `https://eprint.iacr.org/2014/728`.

Papers in Conferences and Workshops with Proceedings

[3] Stefan Krüger, Sarah Nadi, Michael Reif, Karim Ali, Mira Mezini, Eric Bodden, Florian Göpfert, Felix Günther, Christian Weinert, Daniel Demmler, and Ram Kamath. CogniCrypt: Supporting Developers in using Cryptography. In *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*, pages 931–936. IEEE Press, October 2017.

[4] Jacqueline Brendel, Marc Fischlin, Felix Günther, and Christian Janson. PRF-ODH: Relations, Instantiations, and Impossibility Results. In *37th International Cryptology Conference (CRYPTO 2017, Part III)*, volume 10403 of *Lecture Notes in Computer Science*, pages 651–681. Springer, August 2017. Also available as Cryptology ePrint Archive Report 2017/517, `https://eprint.iacr.org/2017/517`.

[5] Felix Günther and Sogol Mazaheri. A Formal Treatment of Multi-key Channels. In *37th International Cryptology Conference (CRYPTO 2017, Part III)*, volume 10403 of *Lecture Notes in Computer Science*, pages 587–618. Springer, August 2017. Also available as Cryptology ePrint Archive Report 2017/501, `https://eprint.iacr.org/2017/501`. **Part of this thesis.**

[6] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT Key Exchange with Full Forward Secrecy. In *36th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2017, Part III)*, volume 10212 of *Lecture Notes in Computer Science*, pages 519–548. Springer, April 2017. Also available as Cryptology ePrint Archive Report 2017/223, `https://eprint.iacr.org/2017/223`.

[7] Marc Fischlin and Felix Günther. Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates. In *2nd IEEE European Symposium on Security and Privacy (EuroS&P 2017)*, pages 60–75. IEEE, April 2017. Also available as Cryptology

ePrint Archive Report 2017/082, https://eprint.iacr.org/2017/082. **Part of this thesis.**

[8] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure Logging Schemes and Certificate Transparency. In *21st European Symposium on Research in Computer Security (ESORICS 2016)*, volume 9879 of *Lecture Notes in Computer Science*, pages 140–158. Springer, September 2016. Also available as Cryptology ePrint Archive Report 2016/452, https://eprint.iacr.org/2016/452.

[9] Marc Fischlin, Felix Günther, Benedikt Schmidt, and Bogdan Warinschi. Key Confirmation in Key Exchange: A Formal Treatment and Implications for TLS 1.3. In *37th IEEE Symposium on Security and Privacy (S&P 2016)*, pages 452–469. IEEE, May 2016. **Part of this thesis.**

[10] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. In *22nd ACM Conference on Computer and Communications Security (CCS 2015)*, pages 1197–1210. ACM, October 2015. Also available as Cryptology ePrint Archive Report 2015/914, https://eprint.iacr.org/2015/914. **Part of this thesis.**

[11] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data Is a Stream: Security of Stream-Based Channels. In *35th International Cryptology Conference (CRYPTO 2015, Part II)*, volume 9216 of *Lecture Notes in Computer Science*, pages 545–564. Springer, August 2015. Also available as Cryptology ePrint Archive Report 2017/1191, https://eprint.iacr.org/2017/1191. **Part of this thesis.**

[12] Felix Günther and Bertram Poettering. Linkable Message Tagging: Solving the Key Distribution Problem of Signature Schemes. In *20th Australasian Conference on Information Security and Privacy (ACISP 2015)*, volume 9144 of *Lecture Notes in Computer Science*, pages 195–212. Springer, June 2015. *Best student paper.* Also available as Cryptology ePrint Archive Report 2014/014, https://eprint.iacr.org/2014/014.

[13] Jean Paul Degabriele, Victoria Fehr, Marc Fischlin, Tommaso Gagliardoni, Felix Günther, Giorgia Azzurra Marson, Arno Mittelbach, and Kenneth G. Paterson. Unpicking PLAID – A Cryptographic Analysis of an ISO-standards-track Authentication Protocol. In *1st International Conference on Research in Security Standardisation (SSR 2014)*, volume 8893 of *Lecture Notes in Computer Science*, pages 1–25. Springer, December 2014. Also available as Cryptology ePrint Archive Report 2014/728, https://eprint.iacr.org/2014/728.

[14] Marc Fischlin and Felix Günther. Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In *21st ACM Conference on Computer and Communications Security (CCS 2014)*, pages 1193–1204. ACM, November 2014. **Part of this thesis.**

[15] Felix Günther, Mark Manulis, and Andreas Peter. Privacy-Enhanced Participatory Sensing with Collusion Resistance and Data Aggregation. In *13th International Conference on Cryptology and Network Security (CANS 2014)*, volume 8813 of *Lecture Notes in Computer Science*, pages 321–336. Springer, October 2014. Also available as Cryptology ePrint Archive Report 2014/382, https://eprint.iacr.org/2014/382.

[16] Nils Fleischhacker, Felix Günther, Franziskus Kiefer, Mark Manulis, and Bertram Poettering. Pseudorandom Signatures. In *8th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2013)*, pages 107–118. ACM, May 2013. Also available as Cryptology ePrint Archive Report 2011/673, https://eprint.iacr.org/2011/673.

[17] Felix Günther, Mark Manulis, and Thorsten Strufe. Key Management in Distributed Online Social Networks. In *12th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2011) / D-SPAN Workshop*, pages 1–7. IEEE, June 2011.

[18] Felix Günther, Mark Manulis, and Thorsten Strufe. Cryptographic Treatment of Private User Profiles. In *15th International Conference on Financial Cryptography and Data Security (FC 2011) / RLCPS Workshop*, volume 7126 of *Lecture Notes in Computer Science*, pages 40–54. Springer, March 2011. Also available as Cryptology ePrint Archive Report 2011/064, `https://eprint.iacr.org/2011/064`.

Papers in Workshops without Proceedings

[19] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A Cryptographic Analysis of the TLS 1.3 draft-10 Full and Pre-shared Key Handshake Protocol. Cryptology ePrint Archive, Report 2016/081, January 2016. `https://eprint.iacr.org/2016/081`. Presented at the TRON (TLS1.3 – Ready or Not?) Workshop / NDSS 2016. **Part of this thesis.**

[20] Pierre-Louis Cayrel, Sidi Mohamed El Yousfi Alaoui, Felix Günther, Gerhard Hoffmann, and Holger Rother. Efficient Implementation of Code-Based Identification Schemes. Technical Report. Presented at the Western European Workshop on Research in Cryptology (WEWoRC 2011), July 2011.

Surveys

[21] Mark Manulis, Nils Fleischhacker, Felix Günther, Franziskus Kiefer, and Bertram Poettering. Group Signatures: Authentication with Privacy. 2012. Survey for the BSI (German Federal Office for Information Security).

Non-Refereed Articles / Articles in Submission

[22] Jacqueline Brendel, Marc Fischlin, and Felix Günther. Breakdown Resilience of Key Exchange Protocols and the Cases of NewHope and TLS 1.3. Cryptology ePrint Archive, Report 2017/1252, December 2017. `https://eprint.iacr.org/2017/1252`.

[23] Matthias Geihs, Oleg Nikiforov, Denise Demirel, Alexander Sauer, Denis Butin, Felix Günther, Gernot Alber, Thomas Walther, and Johannes Buchmann. The Status of Quantum-Based Long-Term Secure Communication over the Internet. 2017.

[24] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data Is a Stream: Security of Stream-Based Channels. Cryptology ePrint Archive, Report 2017/1191, December 2017. `https://eprint.iacr.org/2017/1191`. **Part of this thesis.**

Acknowledgments

Pursuing a Ph.D. is an exciting journey and its success and enjoyment depends on many more people than just the candidate itself. Here, I do want to thank those who made my personal Ph.D. journey possible, successful, and enjoyable.

First and foremost, I am deeply grateful for having Marc Fischlin as my Ph.D. advisor. Marc made it possible that I could join his extraordinary Cryptoplexity research group in Darmstadt, enabling my Ph.D. in the first place. He proved to be an outstanding advisor, both on academic and non-academic matters, and I owe a huge share of what I learned and how I developed during my Ph.D. to him. Working and discussing with Marc, especially one-on-one but also when collaborating with others, was always a great pleasure and his way of challenging results taught me a great deal about the craft of research. Thank you Marc for this opportunity, your trust and support, and for always having an open door and ear for me and my questions.

Joining the Cryptoplexity group was one of the best experiences I could make for my Ph.D. I felt welcome even before my first official day in the group, and since then I have enjoyed being part of an exceptional group of colleagues and friends in and beyond Darmstadt. Thank you, Paul Baecher, Jacqueline Brendel, Chris Brzuska, Özgür Dagdelen, Jean Paul Degabriele, Daniel Demmler, Pooya Farshim, Victoria Fehr, Tommaso Gagliardoni, Patrick Harasser, Christian Janson, Nikolaos Karvelas, Giorgia Azzurra Marson, Sogol Mazaheri, Heike Meissner, Arno Mittelbach, Cristina Onete, Andreas Peter, Bertram Poettering, and Andrea Püchner. Among these, special thanks go to my office mates, namely to Paul for sharing your insights into processes as well as your passion for style and Ti*k*Z, to Giorgia for all the intense research and, more importantly, non-research discussions we had, for traveling with me around the world on truly unforgettable trips, and for being a great friend, and to Christian for many enjoyable chats and your unending support and assurance especially during the last months.

I was fortunate to be able to travel a lot during my Ph.D. and to have the chance of joining a number of groups for shorter and longer research visits. I want to particularly thank Kenny Paterson for being able to visit Royal Holloway in January 2015 together with Giorgia, the focused time he offered back then and the sincere mentoring he has provided me with ever since. Being able to visit QUT in Brisbane in June/July 2015 was an exceptional chance and a great experience that I enjoyed very much and for which I am grateful to Douglas Stebila who has been a wonderful host and mentor. Finally, I want to thank Mihir Bellare for the opportunity to visit UCSD in March 2017, experience San Diego, and enjoy intense research discussions with him.

In the collaboration with others I found research to be most inspiring and enjoyable, and I am thankful for the opportunity to work on many different ideas and topics with great colleagues. Among them, I want to specifically thank Mark Manulis, Andreas Peter, Bertram Poettering, and Thorsten Strufe for giving me the chance to experience research already early in my B.Sc. and M.Sc. studies, and additionally Jacqueline Brendel, Benjamin Dowling, Marc Fischlin, Tibor Jager, Christian Janson, Giorgia Azzurra Marson, Sogol Mazaheri, Arno Mittelbach, Kenny Paterson, and Douglas Stebila for particularly enlightening, inspiring, and engaging collaborations during my Ph.D.

I thank Kenny Paterson for agreeing to co-review my thesis and Johannes Buchmann, Matthias Hollick, and Felix Wolf for joining my defense committee.

Staying connected with a great network of friends at home in Bickenbach and beyond provided me with invaluable support and helped reminding me of the important, non-academic matters in life. I am particularly grateful for numerous friends in the YMCA, enabling and sharing my belief and source of strength and forming a team I am proud to be part of. Among them, special thanks go to Beate and Peter for being great friends and the best outfitters you could wish for, and for hosting me at any times, good or bad. I thank Benedict and Benjamin for our long-standing friendship, curious discussions, and relaxing evenings, Florian for reviving our deep understanding, his care, and the great fun of visiting each other on trips, and Malte for uncountable hours of discussions and most valuable chats, for care and support, and together with Delila and now also Timea for providing me with a space for vacation and rest.

Last, but undoubtedly not least, I am most deeply thankful for the support and love of my family, they ultimately enabled all that I am and so I dedicate this work to them. Among all of them, I particularly thank Jule for all her support throughout, Tabea for noticing me and being there for me, my brother Oliver for being my companion ever since I can remember, and my parents for loving me for who I am. You are the greatest gift in my life.

Felix Günther
Bickenbach, February 2018

My Contribution

Research is, after all, a joint effort in striving for new knowledge, and so are research projects more often than not collaborative work in which colleagues join their intellectual forces. The way the results in this thesis evolved is not different in that regard, and I am grateful that I had the opportunity to work on these results with many great colleagues—thank you Benjamin Dowling, Marc Fischlin, Giorgia Azzurra Marson, Sogol Mazaheri, Kenny Paterson, Benedikt Schmidt, Douglas Stebila, and Bogdan Warinschi.

When the work on a research project is highly collaborative, breaking down and associating particular components of the resulting research paper to individual contributors becomes almost impossible. In the work forming the basis of this thesis, all authors contributed to discussing and formalizing ideas, debating results, and editing their presentation throughout the whole process and resulting paper. This thesis integrates the results from original publications as indicated in the preceding list of publications and listed below in revised, yet often verbatim form, and hence carries joint and sometimes practically indivisible contributions. For the purpose of this thesis, I nevertheless in the following give—where possible—an account of what was specifically my contribution to the results presented in this thesis.

Chapter 4 integrates joint work with Benjamin Dowling, Marc Fischlin, and Douglas Stebila [14, 10, 19, 7].[1] Marc and I [14] jointly devised the core multi-stage key exchange (MSKE) model (in Sections 4.2–4.5) for the analysis of the QUIC protocol (cf. Chapter 5), with me particularly contributing its formalization and the notions of leveled forward secrecy and key independence. I furthermore contributed the compositional result (in Section 4.6). Ben, Marc, Douglas, and I [10, 19] later augmented the MSKE model for our analyses of the TLS 1.3 Diffie–Hellman and pre-shared key handshake modes (cf. Chapter 6). Here, I contributed to the public-key variant and to added technical aspects like concurrent authentication, post-specified peers, or contributive identifiers, while Ben focused on the pre-shared secret variant (MS-PSKE). Finally, Marc and I [7] further extended the MSKE model (resulting in the version presented in this thesis) for our analysis of the TLS 1.3 0-RTT handshakes (cf. Chapter 7), where I contributed formalizing the effects of replays in the model.

Chapter 5 is based on a joint work with Marc Fischlin [14]. Both Marc and I discussed the functionality and high-level security achieved by the QUIC protocol (in Sections 5.2 and 5.3), which I revised for this thesis in the notation of the MSKE model used here. While Marc focused on the Multi-Stage security proof for QUIC, I contributed the Match analysis and the key-independent variant of the QUIC protocol.

Chapter 6 is composed of joint work with Benjamin Dowling, Marc Fischlin, and Douglas Stebila [10, 19]. We all contributed to the functional description of and comments on the TLS 1.3 draft handshakes (in Sections 6.2, 6.4, and 6.7). Ben then focused on the security analysis of the pre-shared key handshake (in Section 6.5) while I focused on the analysis of the full/(EC)DHE handshake and composition (in Sections 6.3 and 6.6).

[1] References in this chapter refer to my list of publications given on pages v ff.

Chapter 7 is based on a joint work with Marc Fischlin [7] with the results presented in this chapter being my contribution.

Chapter 8 is based on a joint work with Marc Fischlin, Benedikt Schmidt, and Bogdan Warinschi [9]. The notions we establish for full and almost-full key confirmation emerged from discussions among all of us, with me particularly contributing to the concept of key-confirmation identifiers (and their binding) and to studying their relationship (in Section 8.2). I furthermore contributed the analysis of key confirmation in TLS 1.3 (in Section 8.3).

Chapters 10 and 11 are based on joint work with Marc Fischlin, Giorgia Azzurra Marson, and Kenny Paterson [11, 24]. I put forward the ideas to study the streaming behavior of channels on both the sender and receiver end (underlying Chapter 10) and of applications aiming to transport atomic messages over a streaming channel (underlying Chapter 11). In Chapter 10, Giorgia and I developed the functional specification and the security properties of confidentiality and integrity of such stream-based channels (in Section 10.2 and 10.3). Giorgia then focused on the relations among the security notions put forward, in particular proving the adapted composition result (in Section 10.3.3). My contribution instead comprises the generic construction of a stream-based channel from AEAD along with its security analysis and comparison with the TLS protocol design (in Section 10.4). In Chapter 11, both Giorgia and I again devised the functional and security definitions for atomic-message channels supporting fragmentation (in Sections 11.2 and 11.3). I then specifically contributed the generic "encode-then-stream" construction and comparative discussion with application's behavior in practice (in Sections 11.1 and 11.4), while Giorgia and I jointly analyzed the construction's security (in Section 11.5).

Chapter 12 is based on a joint work with Sogol Mazaheri [5]. I proposed the idea to study the security of channels deploying a sequence of keys to achieve advanced security properties. Sogol then defined the syntax of such multi-key channels (in Section 12.2) and we jointly devised the framework of security notions capturing confidentiality and integrity with advanced security properties (in Section 12.3). Sogol furthermore established the compositional result while I studied the relations among the security notions we establish (both in Section 12.3.3) as well as proposed and analyzed the generic construction of a multi-key channel including the comparative discussion with TLS 1.3 (in Section 12.4).

Abstract

Secure communication has become an essential ingredient of our daily life. Mostly unnoticed, cryptography is protecting our interactions today when we read emails or do banking over the Internet, withdraw cash at an ATM, or chat with friends on our smartphone. Security in such communication is enabled through two components. First, two parties that wish to communicate securely engage in a *key exchange* protocol in order to establish a shared secret key known only to them. The established key is then used in a follow-up *secure channel* protocol in order to protect the actual data communicated against eavesdropping or malicious modification on the way.

In modern cryptography, security is formalized through abstract mathematical *security models* which describe the considered class of attacks a cryptographic system is supposed to withstand. Such models enable formal reasoning that no attacker can, in reasonable time, break the security of a system assuming the security of its underlying building blocks or that certain mathematical problems are hard to solve. Given that the assumptions made are valid, security proofs in that sense hence rule out a certain class of attackers with well-defined capabilities. In order for such results to be meaningful for the actually deployed cryptographic systems, it is of utmost importance that security models capture the system's behavior and threats faced in that 'real world' as accurately as possible, yet not be overly demanding in order to still allow for efficient constructions. If a security model fails to capture a realistic attack in practice, such an attack remains viable on a cryptographic system despite a proof of security in that model, at worst voiding the system's overall practical security.

In this thesis, we reconsider the established security models for key exchange and secure channel protocols. To this end, we study novel and advanced security aspects that have been introduced in recent designs of some of the most important security protocols deployed, or that escaped a formal treatment so far. We introduce enhanced security models in order to capture these advanced aspects and apply them to analyze the security of major practical key exchange and secure channel protocols, either directly or through comparatively close generic protocol designs.

Key exchange protocols have so far always been understood as establishing a single secret key, and then terminating their operation. This changed in recent practical designs, specifically of Google's QUIC ("Quick UDP Internet Connections") protocol and the upcoming version 1.3 of the Transport Layer Security (TLS) protocol, the latter being the de-facto standard for security protocols. Both protocols derive multiple keys in what we formalize in this thesis as a multi-stage key exchange (MSKE) protocol, with the derived keys potentially depending on each other and differing in cryptographic strength. Our MSKE security model allows us to capture such dependencies and differences between all keys established in a single framework. In this thesis, we apply our model to assess the security of both the QUIC and the TLS 1.3 key exchange design. For QUIC, we are able to confirm the intended overall security but at the same time highlight an undesirable dependency between the two keys QUIC derives. For TLS 1.3, we begin by analyzing the main key exchange mode as well as a reduced resumption mode. Our analysis attests that TLS 1.3 achieves strong security for all keys derived without undesired

dependencies, in particular confirming several of this new TLS version's design goals. We then also compare the QUIC and TLS 1.3 designs with respect to a novel 'zero round-trip time' key exchange mode establishing an initial key with minimal latency, studying how differences in these designs affect the achievable key exchange security. As this thesis' last contribution in the realm of key exchange, we formalize the notion of key confirmation which ensures one party in a key exchange execution that the other party indeed holds the same key. Despite being frequently mentioned in practical protocol specifications, key confirmation was never comprehensively treated so far. In particular, our formalization exposes an inherent, slight difference in the confirmation guarantees both communication partners can obtain and enables us to analyze the key confirmation properties of TLS 1.3.

Secure channels have so far been modeled as protecting a sequence of distinct messages using a single secret key. Our first contribution in the realm of channels originates from the observation that, in practice, secure channel protocols like TLS actually do not allow an application to transmit distinct, or atomic, messages. Instead, they provide applications with a streaming interface to transmit a stream of bits without any inherent demarcation of individual messages. Necessarily, the security guarantees of such an interface differ significantly from those considered in cryptographic models so far. In particular, messages may be fragmented in transport, and the recipient may obtain the sent stream in a different fragmentation, which has in the past led to confusion and practical attacks on major application protocol implementations. In this thesis, we formalize such stream-based channels and introduce corresponding security notions of confidentiality and integrity capturing the inherently increased complexity. We then present a generic construction of a stream-based channel based on authenticated encryption with associated data (AEAD) that achieves the strongest security notions in our model and serves as validation of the similar TLS channel design. We also study the security of such applications whose messages are inherently atomic and which need to safely transport these messages over a streaming, i.e., possibly fragmenting, channel. Formalizing the desired security properties in terms of confidentiality and integrity in such a setting, we investigate and confirm the security of the widely adopted approach to encode the application's messages into the continuous data stream. Finally, we study a novel paradigm employed in the TLS 1.3 channel design, namely to update the keys used to secure a channel during that channel's lifetime in order to strengthen its security. We propose and formalize the notion of multi-key channels deploying such sequences of keys and capture their advanced security properties in a hierarchical framework of confidentiality and integrity notions. We show that our hierarchy of notions naturally connects to the established notions for single-key channels and instantiate its strongest security notions with a generic AEAD-based construction. Being comparatively close to the TLS 1.3 channel protocol, our construction furthermore enables a comparative design discussion.

Zusammenfassung

Sichere Kommunikation ist zu einem essentiellen Bestandteil unseres täglichen Lebens geworden. Weitgehend unbemerkt schützt Kryptographie heute unsere Interaktionen, beispielsweise beim Abrufen von E-Mails oder Online-Banking über das Internet, beim Abheben von Bargeld am Geldautomaten oder beim Chatten mit Freunden auf dem Smartphone. Sicherheit in solcher Kommunikation wird durch zwei Komponenten gewährleistet. Zwei Parteien, die sicher kommunizieren wollen, handeln zunächst in einem *Schlüsselaustausch*-Protokoll einen gemeinsamen geheimen Schlüssel aus. Dieser Schlüssel kann dann im sich anschließenden *sicheren Kanal*-Protokoll verwendet werden, um die eigentlichen zu kommunizierenden Daten gegen Abhören und bösartige Modifikation während des Transports zu schützen.

In der modernen Kryptographie wird Sicherheit durch abstrakte, mathematische *Sicherheitsmodelle* definiert, die die betrachtete Klasse von Angriffen beschreiben, gegen die ein bestimmtes kryptographisches System Schutz bieten soll. Solche Modelle ermöglichen eine formale Argumentation, dass kein Angreifer in vernünftiger Zeit die Sicherheit eines Systems brechen kann, unter der Annahme, dass gewisse zugrundeliegende Komponenten sicher oder bestimmte mathematische Probleme schwierig zu lösen sind. Die Gültigkeit der getroffenen Annahmen vorausgesetzt, schließen Sicherheitsbeweise in diesem Sinn bestimmte Klassen von Angreifern mit wohldefinierten Fähigkeiten aus. Damit entsprechende Resultate auf tatsächlich eingesetzte kryptographische Systeme übertragbar sind, ist es von äußerster Wichtigkeit, dass Sicherheitsmodelle das Verhalten der Systeme und ihre Bedrohungen in der 'realen Welt' so akkurat wie möglich abbilden, jedoch gleichzeitig nicht unnötig fordernd sind, um weiterhin die Konstruktion effizienter Systeme zu erlauben. Scheitert ein Sicherheitsmodell darin, einen in der Praxis realistischen Angriff abzubilden, so bleibt ein solcher Angriff selbst für ein in diesem Modell bewiesen sicheres kryptographisches System eine reale Bedrohung, die im schlimmsten Fall die gesamte Sicherheit des Systems zunichte macht.

In dieser Arbeit ergänzen wir die etablierten Sicherheitsmodelle für Schlüsselaustausch und sichere Kanäle, um fortgeschrittene Sicherheitsaspekte abzudecken, die bislang nicht berücksichtigt oder in neueren Konstruktionen wichtiger praktischer Sicherheitsprotokolle ergänzt wurden. Wir wenden unsere erweiterten Sicherheitsmodelle darüber hinaus an, um die Sicherheit weit verbreiteter Kommunikationsprotokolle direkt oder mittels ähnlicher generischer Konstruktionen zu analysieren.

Nach bisherigem Verständnis etablieren Schlüsselaustausch-Protokolle einen einzigen Schlüssel und sind dann beendet. In neueren Konstruktionen, insbesondere dem QUIC („Quick UDP Internet Connections") Protokoll von Google und der anstehenden Version 1.3 des Transport Layer Security (TLS) Protokolls, hat sich dieses Paradigma allerdings geändert. Beide Protokolle leiten mehrere Schlüssel in einem in dieser Arbeit als mehrstufiger Schlüsselaustausch (MSKE) formalisierten Protokoll ab, wobei die Schlüssel potentiell voneinander abhängig sind und sich in ihrer Stärke unterscheiden. Unsere Arbeit erlaubt es, die Abhängigkeiten und Unterschiede aller dieser Schlüssel in einem einzigen Sicherheitsmodell abzubilden und darin die Sicherheit von QUIC und TLS 1.3 zu analysieren. Für QUIC können wir dabei die erwünschte Gesamtsicherheit bestätigen und gleichzeitig eine unnötige Abhängigkeit zwischen Schlüsseln aufzeigen. Für

TLS 1.3 analysieren wir zunächst den Haupt-Schlüsselaustausch und einen verkürzten Modus, für die wir starke Sicherheitseigenschaften für alle Schlüssel ohne unnötige Abhängigkeiten bestätigen können, in Übereinstimmung mit im Entwurfsprozess von TLS gesetzten Zielen. Wir vergleichen darüber hinaus den Schlüsselaustausch in QUIC und in TLS 1.3 bezüglich eines neuen 'zero round-trip time' Modus zur Vereinbarung eines Schlüssels mit minimaler Latenzzeit und studieren, wie sich Unterschiede zwischen den beiden Entwürfen auf die erreichbare Sicherheit auswirken. Als letzten Beitrag im Bereich Schlüsselaustausch formalisieren wir schließlich eine bislang nicht umfassend formalisierte Schlüsselbestätigungseigenschaft, durch die Verfahren den Kommunikationspartnern die tatsächliche beidseitige Ableitung des Schlüssels zusichern. Unsere Formalisierung verdeutlicht dabei, dass die erreichbare Eigenschaft sich für die beiden Parteien leicht unterscheidet, und erlaubt es uns, die Sicherheit von TLS 1.3 in diesem Bezug zu analysieren.

Die Modellierung sicherer Kanäle hat bislang nur die Absicherung von Sequenzen unteilbarer (atomarer) Nachrichten unter einem einzigen Schlüssel berücksichtigt. Unser erster Beitrag zu Kanälen beruht auf der Beobachtung, dass Kanäle in der Praxis nicht den Transport atomarer Nachrichten, sondern nur das Senden von Datenströmen ohne Kennzeichnung individueller Nachrichten ermöglichen. Notwendigerweise unterscheiden sich dadurch auch die Sicherheitseigenschaften dieser Kanäle von den in bisherigen Modellen definierten, wobei insbesondere die mögliche Fragmentierung von Nachrichten in der Vergangenheit zu Verwirrungen und realen Angriffen auf wichtige Implementierungen von Kanälen geführt hat. In dieser Arbeit formalisieren wir solche strombasierten Kanäle und bilden deren inhärent höhere Komplexität in entsprechend angepassten Sicherheitsdefinitionen für Vertraulichkeit und Integrität ab. Wir geben dann eine generische Konstruktion auf Basis von authentisierter Verschlüsselung mit assoziierten Daten (AEAD) an, die starke Sicherheit erreicht und das Konstruktionsprinzip in TLS bestätigt. Des Weiteren studieren wir die Sicherheit von Anwendungen, die inhärent atomare Nachrichten sicher über einen strombasierten Kanal senden wollen. Auf Basis unserer Formalisierung dieses Sicherheitsziels (in Bezug auf Vertraulichkeit und Integrität) bestätigen wir, dass der weit verbreitete Ansatz, die Nachrichten im zu sendenden Datenstrom zu kodieren, tatsächlich sicher ist. Zuletzt wenden wir uns einem weiteren, neuen Paradigma im Entwurf des TLS 1.3 Kanals zu, nämlich der Möglichkeit, Schlüssel während der Laufzeit eines Kanalprotokolls zu erneuern, um die Sicherheit des Kanals zu erhöhen. Wir schlagen hierzu ein formales Modell für solche Kanäle mit mehreren Schlüsseln vor, welches die fortgeschrittenen Sicherheitsaspekte jenseits von Vertraulichkeit und Integrität in einer Hierarchie abbildet. Wir zeigen, dass sich unsere Hierarchie dabei auf natürliche Weise an die etablierten Definitionen für Kanäle mit nur einem Schlüssel anschließt und instanziieren die stärkste Definition in unserer Hierarchie mit einer generischen, AEAD-basierten Konstruktion, die zudem einen Vergleich mit der ähnlichen TLS 1.3 Kanalkonstruktion ermöglicht.

Contents

Chapter 1

Introduction

Securing the communication between two (or more) people is the foundational goal of cryptography. Early solutions to maintain the secrecy of communication date back as far as 1.900 BC with the first cryptologic hieroglyphs, 475 BC with the Greek "skytale" enciphering tool, or around 50 BC with the Caesar cipher [Kah96]. Today, the vast majority of communication requiring security of one or another form takes place over the Internet. Cryptography—in the form of algorithms implemented as computer programs—consequently secures network connections, e.g., when accessing websites, emails, or the cloud, in securing server-to-server communication between company networks, in mobile communication networks and secure messaging, or in electronic card transactions at ATMs.

Conceptually unaltered throughout history, two core components of a cryptographically secured communication emerged. First, the two communicating parties establish a shared secret and (usually) authenticate each other in what we today call an (authenticated) *key exchange* protocol. In the second step, this key is then used to protect the actual data to be communicated from adversarial eavesdropping or modification in transport (e.g., through enciphering) in what we today call a *secure channel* protocol.

Modern cryptography does not stop at proposing schemes and protocols that intuitively achieve these goals, but aims at grounding their security on cryptographic or mathematical hardness assumptions such as, e.g., (the hardness of) prime factorization or computing discrete logarithms. This is formalized via a complexity-theoretic *reduction* within a formal, mathematical abstraction of the cryptographic system under consideration, a so-called *security model*. Such a security model describes how a potential adversary attacking the system, modeled as a probabilistic polynomial-time Turing machine, may interact with the cryptographic scheme. It then defines under which conditions the adversary, in a specified experiment setup, is considered to have successfully broken the targeted security property of the scheme. A proof of security finally is a complexity-theoretic reduction, transforming a successful adversary against the specified security notion of a cryptographic scheme into a successful adversary against the security of an underlying building block or cryptographic hardness assumption. Assuming the building block's security, resp. cryptographic assumption's hardness, such a reduction asserts that there cannot exists such an adversary capable of breaking the scheme's security with actions considered in the security model.

Naturally, any abstract model of the real world can only account for a limited subset of aspects of this world, and so can a mathematical security model only capture a certain *class* of attack vectors and ways a potential adversary might interact with a cryptographic scheme. Gaps between the attacker capabilities in the real world and those captured in the security model however are dangerous: the security of a cryptographic scheme (and, with it, the information it protects) may in practice be threatened by an adversary capable of mounting a realistic

attack, despite an existing security proof in a model that does not consider the attack in question. Beyond testing the validity and strengths of the cryptographic assumptions relied upon, it is hence of utmost importance to devise security models which capture the power of a potential attacker in the real world as comprehensively and accurately as possible. The aforementioned nature of abstract mathematical models and the impossibility to devise a perfect model inevitably makes this an infinite pursuit striving for better representations of real-world security. This thesis is a contribution to that pursuit in the realm of key exchange and secure channel protocols (in Part I, resp. Part II, of this work), augmenting established security models in both areas in order to capture novel security aspects and recent developments in the design of such protocols.

1.1 Key Exchange

Secure communication is based on a secret, or *cryptographic key*, shared between the two communication partners. While traditionally such secrets had to be exchanged in an out-of-band manner, the availability of public-key cryptography and the seminal protocol to establish a shared symmetric key over an insecure network by Diffie and Hellman [DH76] made it possible to devise protocols that perform a *key exchange* in an online manner right at the start of a communication.

In terms of security, Bellare and Rogaway [BR94] were the first to give a formal treatment of the *notion of security* to be expected from such a key exchange protocol. In their strong security model, an adversary interacts with an arbitrary number of protocol executions, controls the whole communication network (able to eavesdrop on, manipulate, or drop any message[2]), is able to corrupt some parties in the system (learning their long-term secrets), and is allowed to reveal the keys derived in some of the protocol runs. Still, any key established in an uncompromised protocol execution should look like a completely random string to such an adversary and the involved parties should authenticate correctly. This yields the intuitive guarantee that, to an adversary as described, a key established via the key exchange protocol is as secret as if it has been established in an authentic out-of-band manner. The Bellare–Rogaway model, which we recap in Chapter 3, has become the cryptographic de-facto standard for security analyses of proposed and deployed protocols as well as the foundation for extensions to capture further security properties.

1.1.1 Multi-Stage Key Exchange

In the first part of this thesis, our first contribution also is an extension of the Bellare–Rogaway model in order to capture novel paradigms exhibited in recent key exchange protocol designs. Classically, starting from Bellare and Rogaway [BR94] and kept throughout subsequent formalizations, a key exchange protocol is understood to be executed in order to establish a (single) shared key and then terminate. In recent designs of real-world key exchange protocols this paradigm changed, particularly with the QUIC ("Quick UDP Internet Connections") protocol [QUI] proposed by Google in 2013 and by now globally deployed [LRW+17], and the upcoming next version 1.3 of the de-facto standard security protocol on the Internet, the Transport Layer Security (TLS) protocol [DR08, Res18], but also in key exchange designs underlying secure messaging protocols like Signal [Sig]. These protocols interleave the key establishment and communication in a continuous process deriving multiple keys, used for the actual communication

[2]The adversary's omnipotence in fully controlling the communication network resembles the Dolev-Yao adversary model [DY83]. In this thesis, we however work in the computational setting and hence furthermore allow the adversary to tamper arbitrarily with the messages exchanged, not restricting it to an abstract, symbolic or algebraic representation (see, e.g., Abadi and Rogaway [AR02] for a comparative discussion).

or other purposes like exporting additional key material to an application. Established security models cannot capture such designs in their full extent as they can only focus on one of the keys established and are unable to, e.g., describe the mutual effects compromises of a subset of the multiple keys established may have on their security.

To overcome this gap, we introduce in Chapter 4 a security model for *multi-stage key exchange (MSKE)* protocols which is capable of capturing the security and dependency of all keys derived, at potentially varying security levels, in a single security framework. Our model in particular allows to capture the effects of compromises of different secrets (long-term and medium-lived) as well as inter-dependencies and varying authentication levels of keys derived at different stages in the key exchange. It moreover can treat both protocols with symmetric-key and with public-key long-term secrets as well as the effects of possibilities to replay messages in some key exchange designs aiming at low-latency key exchange. These features enable the model to comprehensively reflect the relevant core cryptographic components of both the QUIC key exchange protocol as well as the three different key exchange modes of TLS 1.3.

Our security model is further accompanied by a compositional result that establishes sufficient conditions under which the keys established in a multi-stage key exchange protocol can safely be used in a follow-up symmetric-key protocol. This result allows to argue the joint security of a secure MSKE protocol and, e.g., the subsequent secure channel protocol and hence reduces analysis complexity by enabling an independent and modular security analyses of both cryptographic components.

1.1.2 The QUIC Protocol

We first apply our MSKE security model, in Chapter 5, to analyze the security of Google's QUIC protocol [QUI]. QUIC was introduced as a secure connection protocol (comprising both cryptographic key exchange and channel components) that establishes connections with low latency while maintaining the security guarantees of the established TLS protocol [DR08]. In the meantime, it has been deployed at large scale in Google's service infrastructure and the Chrome browser [LRW+17], and sparked an IETF working group [QWG] aiming to provide a standards-track specification for QUIC. The QUIC protocol particularly aims at reducing the round complexity of the key exchange, i.e., the number of times messages have to be sent back and forth between the two communicating parties. To this end, it introduces a so-called zero round-trip time (0-RTT) key exchange mode. This mode enables a client to immediately send data along with its first key exchange message to a server it previously communicated with, hence drastically reducing the initialization delay of the secure connection. The 0-RTT data is encrypted under an initial key; both parties then update to a stronger main key with the reply of the server.

QUIC is a particularly interesting example to study in the MSKE model. Most notably, it establishes two keys within one protocol execution, making it a multi-stage key exchange protocol in the first place. These keys are moreover crucially intertwined in terms of security, with the first key authenticating the second and hence compromising the former before the latter is established leads to a security break—an insight which could not be formally captured in previous security models. We can furthermore establish in our model that the second key achieves stronger secrecy than the first, remaining secure even if the involved long-term secrets are later compromised (so-called forward secrecy).

1.1.3 The TLS 1.3 Protocol: Diffie–Hellman and Pre-shard Keys

The remaining chapters in this part of the thesis are concerned with the upcoming next version of the Transport Layer Security (TLS) protocol [DR08], TLS 1.3 [Res18]. The TLS protocol

constitutes the de-facto standard for secure application-level communication over the Internet, among many things protecting the security of billions of web, e-mail, and cloud accesses every day. TLS comprises both a key exchange component, the so-called *handshake protocol* that allows a client and a server to authenticate each other and to establish a key, as well as a subsequent secure channel component, the *record layer protocol* providing confidentiality and integrity for communication of application data. A security protocol as widely deployed as TLS is a particularly exposed target for attacks, and so it is perhaps not surprising that numerous successful attacks on different TLS versions have been found over the past years. Practical attacks that have received significant attention include exploiting weaknesses in cryptographic components (like RC4 [ABP+13], hash functions [BL16b], or 64-bit block ciphers [BL16a]), flaws in the protocol design (e.g., BEAST [Duo11], the Lucky 13 attack [AP13], the triple handshake attack [BDF+14], the POODLE attack [MDK14], or the Logjam attack [ABD+15]), or flaws in implementations (such as the Heartbleed [Cod14], SMACK [BBD+15], Cloudbleed [Orm17], or ROBOT [BSY17] attack). In parts to overcome structural weaknesses underlying the above attacks but also and equally important to add desired functional and privacy features the Internet Engineering Task Force (IETF) currently devises a *new TLS standard, TLS 1.3* in a series of drafts. These new features include a low-latency handshake mode (as in QUIC), deriving intermediate keys to encrypt parts of the handshake for privacy, or deploying a sequence of keys in the channel protocol for advanced security. In particular, TLS 1.3 hence is a *multi-stage* key exchange protocol for which our MSKE security model enables a comprehensive analysis.

We begin our analysis of the TLS 1.3 protocol drafts in Chapter 6 by establishing MSKE security of the main *Diffie–Hellman-based* handshake and the abbreviated handshake for resuming previous connections based on *pre-shared keys*. Both handshakes establish several keys, in parts used within the key exchange, to protect communication, subsequent resumption handshakes, or external cryptographic applications, and we capture them as distinct stages within our model. This enables us to capture their individual security properties, e.g., the intermediate key protecting parts of the handshake messages having a different authentication level than the other keys established. The generic composition result for the MSKE model from Chapter 4 furthermore enables us to argue security of using the established handshake keys, e.g., in the subsequent record protocol. Our analysis finally provides insights into the cryptographic choices made during the design process of TLS 1.3 (to which our analysis itself contributed as well), which we comment on at the end of the chapter.

1.1.4 The TLS 1.3 Protocol: Zero Round-Trip Time and Replays

In Chapter 7 we turn to the third of the handshake modes specified in TLS 1.3 drafts, namely a *low-latency (0-RTT)* mode combined with either a Diffie–Hellman or a pre-shared key handshake. The 0-RTT mode allows a client to immediately establish a shared key with a server in order to start communicating securely already along with its first handshake message, saving substantially on the initial communication delay. Both parties then immediately continue with an intertwined Diffie–Hellman or pre-shared key handshake to establish further (and more secure) keys within the same overall handshake.

We first discuss a security issue that necessarily arises in 0-RTT handshakes, namely that 0-RTT messages can be *replayed* by a man-in-the-middle adversary due to the lack of contribution from both communication parties, and how these issues are handled in different variants of the QUIC and TLS 1.3 protocols. As we will see, TLS 1.3 accepts 0-RTT replays as generally inevitable while the original QUIC design aimed at preventing them at least on the key exchange level. We therefore leverage the ability of the MSKE model to distinguish between replayable and non-replayable keys (or stages) and their differences in security. In our model, we then analyze both a Diffie–Hellman-based 0-RTT handshake draft and one based on pre-shared keys.

We establish that both variants constitute a secure multi-stage key exchange protocol and that, most importantly, the added 0-RTT mode achieves the expected security and does not negatively affect the security of the main handshake keys derived. We provide a comparative discussion of both TLS 1.3 0-RTT mode variants as well as the QUIC design.

1.1.5 The TLS 1.3 Protocol: A Formal Treatment of Key Confirmation

As our last contribution concerning the TLS 1.3 handshake protocol, we finally consider an additional property of key exchange protocols, *key confirmation*, that has not been comprehensively captured in previous key exchange models, despite being prominently mentioned by practitioners in many protocol design specifications, including that of TLS [DR08, Res18]. Key confirmation intuitively is the property that, when one party accepts with a key in a key exchange execution, it is ensured that the other party also accepts with the same key. While for security (in the style of Bellare–Rogaway [BR94]) the assurance that no third party can learn the established key is sufficient, key confirmation provides an additional functional guarantee ensuring, e.g., that an application does not waste resources on sending data to a peer that did not actually accept the shared key.

In Chapter 8, we propose a comprehensive formalization of key confirmation in key exchange protocols in order to enable a formal treatment of this property and discussion of how it can be, resp. is, achieved generically and in existing protocol designs. Our model in particular exposes subtle differences in the key-confirmation guarantees the two parties running the key exchange can expect. The party accepting last can indeed be assured of (full) key confirmation in the sense that the other party *already accepted* with the same key. The party accepting first however can at most obtain the (almost-full) key confirmation guarantee that the other party will derive the same key *in case it accepts* after obtaining the final key exchange messages. We apply our model to the main TLS 1.3 handshake in which, as in previous TLS versions, a `Finished` message ensures key confirmation through a message authentication code over the full communication transcript. Interestingly, our analysis exhibits that already a shortened variant of the handshake *without* `Finished` messages does provide the same optimal key confirmation guarantees, giving insights in the potential misconception that added message authentication codes are always necessary to achieve key confirmation.

1.2 Secure Channels

Having established a shared secret key for their communication, both parties then execute a secure channel protocol in order to securely transmit the actual communication data. In this setting, 'securely' refers to such data being protected from both passive eavesdropping as well as manipulation through active adversaries; the targeted security goals are hence *confidentiality* and *integrity*, and the basic underlying cryptographic tool is that of *(symmetric-key) encryption*.

Formalizing *security notions* for confidentiality and (later) integrity constitutes further foundational work in modern cryptography originating from Goldwasser and Micali [GM84] followed by Naor and Yung [NY90], Rackoff and Simon [RS92], Bellare and Namprempre [BN00], and Bellare and Rogaway [BR00], as we will recap in Chapter 9. Today, the building block of *authenticated encryption (with associated data)* (AEAD) due to Rogaway [Rog02] combines both confidentiality and integrity guarantees for symmetric encryption of individual messages in a single and highly efficient component (e.g., AES-GCM [Dwo07]). Cryptographically secure channel protocols in practice (like the Transport Layer Security (TLS) protocol [DR08] or the Secure Shell (SSH) protocol [YL06a]) of course are used to transmit more than a single (application) message only. In a sequence of messages integrity should hence, beyond the integrity of single messages on their own, also be concerned with protecting against reordering, replays,

or dropping of messages. Bellare, Kohno, and Namprempre [BKN02, BKN04] recognized that formalizing *stateful* encryption (and decryption) operations and according security notions are necessary to capture such security properties on sequences of messages in their analysis of the SSH protocol. Their notions for *stateful authenticated encryption* are widely accepted as the right formalization of secure cryptographic channels and have been extended in a number of works (see Section 1.3 below) and used to analyze the security of practical channel protocols, including SSH [BKN02, BKN04] and TLS [PRS11, JKSS12, KPW13].

Still, the model of Bellare, Kohno, and Namprempre [BKN04] exhibits limitations in capturing certain structural aspects of channels in practice, which specifically in the case of the SSH protocol has led to a mismatch between the security guarantees assured in their model and the expected practical security. More precisely, Albrecht, Paterson, and Watson [APW09] demonstrated a plaintext recovery attack breaking the basic confidentiality protection in the SSH protocol which exploits the processing of *fragmented* ciphertexts in SSH. This attack vector was not reflected in [BKN04] (and hence does not contradict their security result) as here messages and ciphertext are considered to be *atomic* (i.e., undividable) blocks, not allowing an adversary to submit fragmented ciphertexts. Boldyreva et al. [BDPS12] extended the model of [BKN04] in order to reflect such ciphertext fragmentation (and, thereby, capture the above attack).

1.2.1 Stream-Based Channels

All formalizations of secure channels so far (including [BDPS12]) however still treat messages as atomic objects. This is in contrast to the behavior of channels in practice, as the interface provided to applications by most channel protocols, both non-secured as the Transmission Control Protocol (TCP) [Pos81b] as well as the most relevant secure channels protocols (including TLS [DR08], SSH [YL06a], or QUIC [QUI]), is a *streaming* one: applications submit (fragments of) a continuous *data stream* of bits (or bytes) to be transmitted, which the channel protocol splits up—usually without control of the application—into chunks to be encrypted and transferred. In general, channels in practice thus do not preserve the boundaries of application messages, providing applications with an interface quite distinct from that considered in cryptographic literature so far.

In Chapter 10 we approach this gap in the formalism of secure channels by introducing the notion of *stream-based channels* in order to more accurately model the streaming interface provided by channels in practice. Lacking the structure of a one-to-one correspondence between (atomic) messages and ciphertexts, the security notions we establish turn out to be considerably more involved than the ones for stateful encryption. We build upon the work of Boldyreva et al. [BDPS12] for confidentiality treating ciphertext fragmentation and additionally consider fragmentation of messages as well as integrity. The latter allows us to lift the generic composition theorem for symmetric encryption by Bellare and Namprempre [BN00] to the streaming setting, namely that weak confidentiality against passive adversaries together with integrity of ciphertexts yields strong confidentiality against active attacks. We finally provide a generic construction of a stream-based channel from AEAD achieving strong confidentiality and integrity guarantees. Being structurally close to the TLS record protocol our construction furthermore enables a discussion of that protocol's design.

1.2.2 Atomic-Message Channels Supporting Fragmentation

The interface provided by many real-world channel protocols that we capture in our notions of stream-based channels in Chapter 10 suits well many applications that desire a streaming interface, e.g., for transferring large files or video streaming. Other application layer protocols

are however actually *message-based*, e.g., in chat settings or the transmission of HTTP [FGM+97] headers. Their security may crucially rely upon messages being processed as atomic blocks, and practical examples including TLS truncation attacks [SP13] and the 'cookie-cutter' attack [BDF+14] demonstrate that (mis)interpreting partial, fragmented messages can have disastrous consequences for security in such applications.

In Chapter 11 we therefore study how applications can safely transport *atomic messages* over a *stream-based channel*, formalizing the resulting concept and security for *atomic-message channels supporting fragmentation*. Extending the work of Boldyreva et al. [BDPS12] in particular by integrating a notion of integrity, our framework allows us to capture the common approach of applications to encode their atomic messages within the bit stream to be sent in the secure stream-based channel. We generalize this approach in what we call the 'encode-then-stream' construction and show that it indeed lifts confidentiality and integrity guarantees from the stream-based channel to the (atomic-message) application interface under reasonable assumptions. Our construction also casts a formal light on the potential misunderstanding of security guarantees provided by stream-based and atomic-message channels that led to the above-mentioned truncation attacks [SP13, BDF+14].

1.2.3 Multi-key Channels

In the last chapter of this thesis, we turn towards a novel feature of the upcoming next version of the Transport Layer Security protocol, TLS 1.3 [Res18]. The TLS 1.3 record protocol includes a key-updating mechanism that allows parties to deploy a sequence of multiple keys for encryption instead of a single, fixed key. Such key updates were introduced for both functional reasons (namely that long-lived TLS connections may exceed the limits of how much data can safely be encrypted under a single key) and to enhance the channel's security (especially providing forward security of communication within a channel prior to a compromise of th currently used key).

All models for secure channels so far consider only a single, fixed key as the source of cryptographic protection. In Chapter 12, we introduce the first formalization of a *multi-key channel* that deploys a sequence of multiple keys, each constituting a *phase*. We provide a modular framework of security notions that beyond reflecting the basic notions of confidentiality and integrity in the multi-key setting also capture the two advanced security goals aimed at with multi-key channels, which we denote *forward security* and *phase-key insulation*. Forward security transfers the well-established concept (e.g., in key exchange protocols) that a full compromise of key material at some point in time should not endanger the security of prior communication. Phase-key insulation addresses the more fine-grained compromise of an encryption key in a specific phase of the channel and demands that the security in other, uncompromised phases is still maintained. We study the relations between the various notions in the hierarchy spanned by our framework, in particular establishing forward secrecy and phase-key independence as independent notions as well as that our notions when restricted to a single key naturally connect to the established notions for stateful encryption. We finally provide an instantiation of our strongest confidentiality and integrity notions through a generic construction from AEAD and pseudorandom functions which reflects aspects of the TLS 1.3 record protocol and thus enables a review that design.

1.3 Related Work

In the following, we discuss (further) work preliminary or related to the results presented in this thesis, accounting for the most relevant giants' shoulders this work stands on.

Key exchange models. Following the seminal work by Bellare and Rogaway [BR94], a substantial body of work extended their concept and formalism of a secure key exchange protocol. For example, Blake-Wilson et al. [BM97, BWJM97] extended the symmetric-key treatment of Bellare and Rogaway to the public-key setting; we integrate both settings in our MSKE model (in Chapter 4). Bellare and Rogaway covered the settings of three-party protocols [BR95] and, together with Pointcheval, that of password-based key exchange [BPR00], in particular establishing different concepts for identifying partnered sessions jointly running the key exchange protocol, including the notion of session identifiers we use for the key exchange models in this work, and treating forward secrecy [Gün90, DVOW92]. Shoup [Sho99] established notions for key exchange security in the simulation-based setting. Canetti and Krawzyck [CK01] and La Maccia et al. [LLM07] considered extended compromise settings including the leakage of ephemeral secrets or state; we adopt their approach in capturing compromises of medium-lived secrets in the MSKE model. Cohn-Gordon et al. [CGCG16] recently also considered recovery after compromises. Using the secret established in a key exchange protocol within a subsequent symmetric-key protocol (e.g., a channel) is the prime goal of running the key exchange in the first place. Composability of key exchange protocols with subsequent protocols has been studied by Canetti and Krawczyk [CK02b] and later Küsters et al. [KT11, KR17] in the Universal Composability (UC) framework [Can00], and by Brzuska et al. [BFWW11, BFS+13] in the game-based setting; the latter forming the basis for our compositional results in the MSKE model. To overcome obstacles in analyzing the TLS handshake in established key exchange models, Jager et al. [JKSS12] introduced in a monolithic model for authenticated and confidential channel establishment (ACCE) integrating the key exchange and channel phases. Further security models focusing on specific real-world aspects of key exchange protocols include the treatment of certification systems [BCF+13], capturing downgrade resilience [BBF+16], or tailored models for zero round-trip time key exchange [HJLS17] or secure messaging protocols [CGCD+17, BSJ+17, CGCG+17], the latter building upon the MSKE model. Our security models for multi-stage key exchange (in Chapter 4) and key confirmation (in Chapter 8) belong to this line of research extending the established core of key exchange models towards advanced, previously unexplored security properties aimed at in practice.

Analyses of QUIC. In an independent and concurrent work to our analysis of Google's QUIC protocol [FG14] presented in Chapter 5, Lychev et al. [LJBN15] also investigated the security of QUIC in an extension of the ACCE model [JKSS12] and gave a security proof based on similar assumptions. Their work was later revisited in an automated verification analysis by Sakurada et al. [SYHY16] using the ProVerif tool [Pro]. Langley et al. [LRW+17] gave an account of the experience from their Internet-scale deployment of QUIC which we also refer to for further information on non-cryptographic aspects of QUIC.

Analyses of TLS. As one of the most important and widely used security protocols, the Transport Layer Security (TLS) protocol attracted substantial efforts not only in attacks but also in analyzing and understanding its security. Formal results were first obtained for truncated variants of the handshake [MSW08, GMP+08]. The first full handshake mode (of TLS 1.2) was analyzed by Jager et al. [JKSS12] in their ACCE model combining key exchange and channel security. Subsequent analyses include providing results for further handshake modes and ciphersuites [KPW13, KSS13, LSY+14], the interaction of multiple handshake runs in TLS renegotiation [GKS13], ciphersuite and version negotiation in TLS [DS15], or exporting key material from a TLS connection [BJS16]. The miTLS [miT] team significantly contributed to understanding the security and weaknesses of TLS implementations with their verified miTLS

reference implementation [BFK+13], in particular providing a security analysis of the TLS 1.2 handshake within their implementation [BFK+14].

While all analyses of TLS 1.2 were conducted only after that protocol version was specified (in 2008), the development process for the TLS 1.3 protocol was accompanied by a significant academic effort to assess the security of its draft versions while they were being specified. Work on the first drafts of TLS 1.3 started in April 2014. Through 24 draft versions (which we refer to as `draft-xx` throughout this thesis, with `xx` indicating the draft number), TLS 1.3 by now reached a relatively stable state in terms of its cryptographic design, the latest draft as of February 2018 being `draft-24` [Res18]. We refer to Paterson and van der Merwe [PvdM16] for a comprehensive treatment of this design process but list its most notable academic contributions in the following, excluding our analyses of the multi-stage security and key confirmation properties of the TLS 1.3 handshake protocol drafts which we present in Chapters 6–8. Using a constructive-cryptography approach [MR11], Kohlweiss et al. [KMO+14] studied the handshake of `draft-05` [Res15b] and Badertscher et al. [BMM+15] the record protocol of `draft-08` [Res15d]. In `draft-07` [Res15c], the handshake's cryptographic core was substantially reworked based on the modular OPTLS design by Krawzyck and Wee [KW16], elegantly integrating the main Diffie–Hellman, resumption, and 0-RTT modes within a uniform handshake and key schedule structure. Krawczyk [Kra16b] also provided a formal treatment of the post-handshake client authentication introduced in TLS 1.3. Cremers et al. [CHSvdM16, CHH+17] conducted a comprehensive symbolic analysis of several TLS 1.3 drafts starting from `draft-10` [Res15e] based on the Tamarin tool [SMCB12], covering all specified handshake modes. Li et al. [LXZ+16] studied `draft-10` in a multi-level extension of the multi-stage key exchange model we present in Chapter 4 to capture interaction of different handshake modes. Further symbolic analyses were performed using the ProVerif [Bla16] tool on `draft-11` [Res15f] by the CELLOS (Cryptographic protocol Evaluation toward Long-Lived Outstanding Security) team [Mat16] as well as on `draft-18` [Res16d] by Bhargavan et al. [BBK17]. The miTLS team extended their reference implementation, in particular assessing the `draft-18` record protocol in [DLFK+17]. The interaction between different TLS versions (as well as QUIC) was studied in terms of weaknesses in the PKCS#1 v1.5 encryption [Kal98] by Jager et al. [JSS15] and the downgrade resilience of TLS 1.3 was comprehensively analyzed by Bhargavan et al. [BBF+16]. Overall, the community effort including both academic and industry contributions was well-received by the IETF TLS working group, acknowledging and incorporating the provided feedback in the TLS 1.3 design process (cf. again Paterson and van der Merwe [PvdM16] as well as the security overview in the draft standard [Res18, Appendix E]).

Secure Channels. The first formalization of cryptographically secure channels by Bellare, Kohno, and Namprempre [BKN02, BKN04] spawned a line of research studying and extending the models for channels in various directions. For example, Kohno, Palacio, and Black [KPB03] introduce a hierarchy of channel notions capturing different protection levels against replays, reordering, and dropping of messages. Their hierarchy was studied further for authentication and AEAD schemes by Boyd et al. [BHMS16] who provide generic transformations between the hierarchy levels and apply them in an analysis of the TLS 1.2 record protocol. In simulation-based settings, Shoup [Sho99] gave a basic functional model for secure channels, Canetti established a concept of secure channels in the UC framework [Can00], and Canetti and Krawczyk [CK01] introduced composable notions of channel security which Namprempre [Nam02] then characterized in terms of standard, game-based notions for authenticated encryption. In the constructive-cryptography setting, Maurer and Tackmann [MT10] formalized channel security in terms of transformations from encryption and authentication; Badertscher et al. [BMM+15] later defined a constructive-cryptography counterpart of a stateful AEAD scheme as abstraction of a secure channel. In the course of studying the security of TLS, Jager et al. [JKSS12]

introduced the monolithic ACCE model combining key exchange and channel phases, with the latter building upon a work by Paterson et al. [PRS11] capturing the padding mechanism in TLS as length-hiding authenticated encryption. Reflecting implicit information leakage through behavioral differences in processing erroneous ciphertexts, Boldyreva et al. [BDPS14] considered channels that distinguish multiple decryption errors and their effects on the generic composition result by Bellare and Namprempre [BN00]. The same authors also considered the effects of fragmentation of ciphertexts on confidentiality for secure channels [BDPS12], forming a starting point for our notions of stream-based channels from [FGMP15] and of atomic-message channels supporting fragmentation from [FGMP17] presented in Chapters 10 and 11, later and concurrently to [FGMP15, FGMP17] augmented with integrity notions by Albrecht et al. [ADHP16]. Notions approaching the streaming behavior of channels have also been developed by Bhargavan et al. [BFK$^+$13, DLFK$^+$17] as part of their verified implementation analyses of the TLS record protocol. Their security models accompanying the verified implementation also informally touch upon the security implications of deploying a sequence of multiple keys in the TLS 1.3 channel which we capture formally in our general model for multi-key channels in Chapter 12. Boyd and Hale [BH17] studied security notions for secure termination of channels with an application to TLS. Giving a characterization of how unidirectional channels are combined in practice, Marson and Poettering [MP17a, MP17b] studied the security of bidirectional channels and channel security in group communication.

Chapter 2

Preliminaries

In this chapter, we introduce the basic notation used throughout this thesis and recap some fundamental concepts and general cryptographic building blocks. Preliminaries specifically required in the key exchange and secure channels settings are provided in Chapters 3 and 9, respectively.

2.1 Notation

We begin by establishing some of the common notation used within this thesis. We denote by $\mathbb{N}$ the natural numbers as the set of non-negative integers, by $\mathbb{R}$ the real numbers, and by $\emptyset$ the empty set. We write a bit as $b \in \{0,1\}$ and a (bit) string as $s \in \{0,1\}^*$ with $|s|$ indicating its (binary) length; further $\{0,1\}^n$ for the set of bit strings of length n. Given two bit strings $s, t \in \{0,1\}^*$ we denote by $s\|t$ their concatenation and by $s \oplus t$ their bitwise XOR (for $|s| = |t|$). We write $x \leftarrow y$ for the assignment of value y to the variable x and $x \xleftarrow{\$} X$ for uniformly sampling x from the (finite) set X.

For an algorithm $\mathcal{A}$ we write $x \leftarrow \mathcal{A}(y)$ or $\mathcal{A}(y) \rightarrow x$, resp. $x \xleftarrow{\$} \mathcal{A}(y)$ or $\mathcal{A}(y) \xrightarrow{\$} x$, for the algorithm deterministically, resp. probabilistically, outputting x on input y. We indicate by $\mathcal{A}^{\mathcal{O}}$ an algorithm $\mathcal{A}$ running with oracle access to some other algorithm $\mathcal{O}$. When defining security in the asymptotic setting, we indicate by λ the security parameter and provide it (implicitly) as $\mathcal{A}(1^\lambda)$ in unary representation 1^λ to an algorithm $\mathcal{A}$. We call an algorithm efficient if it runs in polynomial time in λ as a (probabilistic) Turing machine (i.e., if there exists a polynomial p such that $\mathcal{A}$ takes at most $p(\lambda)$ steps on input of length λ); we then also say $\mathcal{A}$ is a probabilistic polynomial-time (PPT) algorithm. We say that a function is efficient if it can be computed by an efficient algorithm. We call a function $f\colon \mathbb{N} \rightarrow \mathbb{R}$ negligible if for all positive polynomials p there exists an $N \in \mathbb{N}$ such that for all $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.

2.2 Cryptographic Building Blocks and Assumptions

In the following, we recap some basic and well-studied cryptographic building blocks deployed in the key exchange and secure channel protocols covered in this thesis. For a detailed discussion beyond the common formalization of their syntax and security we refer to, e.g., Katz and Lindell [KL08].

2.2.1 Pseudorandom Functions

A pseudorandom function (PRF) is a keyed function mapping inputs to a (pseudo)random-looking output which is indistinguishable from the output of a truly random function. PRFs

find application in key derivation steps of key exchange protocols as well as (multi-key) channels. We define their security as follows [KL08].

Definition 2.1 (Pseudorandom function and PRF security)**.** *Let* $f\colon \{0,1\}^* \times \{0,1\}^{i(\lambda)} \to \{0,1\}^{o(\lambda)}$ *be an efficient keyed function having input and output length* $i(\lambda)$*, resp.* $o(\lambda)$*, when keyed with* $k \in \{0,1\}^\lambda$*. We say that* f *is* pseudorandom *if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{f,\mathcal{A}} := \left|\Pr\left[\mathcal{A}^{f(k,\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{g(\cdot)}(1^\lambda) = 1\right]\right|,$$

where the PRF key $k \xleftarrow{\$} \{0,1\}^\lambda$ *is sampled uniformly at random and* g *is randomly chosen from the set of all functions* $\{0,1\}^{i(\lambda)} \to \{0,1\}^{o(\lambda)}$*.*

2.2.2 Message Authentication Codes

Message authentication codes (MACs) enable two parties holding a secret key to authenticate messages through a MAC (tag) value, as well as to verify the authenticity of such tags on messages. In key exchange protocols, MACs are used in order to authenticate parties in a setting with shared secret keys and to attest the integrity of the (transcript of) messages exchanged. We state the syntax for MAC schemes and their standard security notions of existential and strong unforgeability under chosen-message attacks [KL08].

Definition 2.2 (Message authentication code scheme)**.** *A* message authentication code (MAC) *scheme* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{Verify})$ *consists of three efficient algorithms defined as follows.*

- $\mathsf{KGen}(1^\lambda) \xrightarrow{\$} K$*. On input a security parameter* 1^λ*, this probabilistic algorithm outputs a MAC key* K*.*
- $\mathsf{Tag}(K, m) \xrightarrow{\$} \tau$*. On input a key* K *and a message* $m \in \{0,1\}^*$*, this (possibly) probabilistic algorithm outputs a MAC tag* τ*.*
- $\mathsf{Verify}(K, m, \tau) \to \{0,1\}$*. On input a key* K*, a message* m*, and a tag* τ*, this deterministic algorithm outputs* 1 *(indicating validity of the tag) or* 0 *(otherwise).*

Definition 2.3 (Existential and strong unforgeability of MACs)**.** *Let* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{Verify})$ *be a MAC scheme and experiments* $\mathsf{Expt}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}}(1^\lambda)$ *and* $\mathsf{Expt}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 2.1.*

We say that MAC *provides* existential (resp. strong) unforgeability under chosen-message attacks *(*EUF-CMA*, resp.* SUF-CMA*) if for all PPT adversaries the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}} := \Pr\left[\mathsf{Expt}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}}(1^\lambda) = 1\right], \quad \textit{resp.} \quad \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}} := \Pr\left[\mathsf{Expt}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}}(1^\lambda) = 1\right].$$

2.2.3 Signatures

(Digital) signatures allow a signer holding a secret key (and only the signer) to issue a publicly verifiable signature authenticating a message. They are in particular used in key exchange protocols to authenticate parties in a public-key setting and the exchanged communication transcript. We recap the syntax of digital signatures together with their standard security notions of existential and strong unforgeability under chosen-message attacks [KL08, Kat10].

Definition 2.4 (Signature scheme)**.** *A* signature scheme $\mathsf{Sig} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ *consists of three efficient algorithms defined as follows.*

$\underline{\mathsf{Expt}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}}(1^\lambda)}$:
1 $K \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
2 $Q \leftarrow \emptyset$
3 $(m^*, \tau^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Tag}}}(1^\lambda)$
4 return 1 iff $(m^*, *) \notin Q$ and $\mathsf{Verify}(K, m^*, \tau^*) = 1$

$\underline{\mathsf{Expt}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}}(1^\lambda)}$:
1 $K \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
2 $Q \leftarrow \emptyset$
3 $(m^*, \tau^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Tag}}}(1^\lambda)$
4 return 1 iff $(m^*, \tau^*) \notin Q$ and $\mathsf{Verify}(K, m^*, \tau^*) = 1$

$\underline{\mathcal{O}_{\mathsf{Tag}}(m)}$:
1 $\tau \xleftarrow{\$} \mathsf{Tag}(K, m)$
2 $Q \leftarrow Q \cup \{(m, \tau)\}$
3 return τ

$\underline{\mathsf{Expt}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{A}}(1^\lambda)}$:
1 $(sk, pk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
2 $Q \leftarrow \emptyset$
3 $(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(1^\lambda, pk)$
4 return 1 iff $(m^*, *) \notin Q$ and $\mathsf{Verify}(pk, m^*, \sigma^*) = 1$

$\underline{\mathsf{Expt}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{A}}(1^\lambda)}$:
1 $(sk, pk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
2 $Q \leftarrow \emptyset$
3 $(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(1^\lambda, pk)$
4 return 1 iff $(m^*, \sigma^*) \notin Q$ and $\mathsf{Verify}(pk, m^*, \sigma^*) = 1$

$\underline{\mathcal{O}_{\mathsf{Sign}}(m)}$:
1 $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$
2 $Q \leftarrow Q \cup \{(m, \sigma)\}$
3 return σ

Figure 2.1: Security experiments for *existential and strong unforgeability under chosen-message attacks* (EUF-CMA, resp. SUF-CMA) for MAC and signature schemes. We write $(a, *) \notin Q$ if $\nexists b$ s.t. $(a, b) \in Q$.

- $\mathsf{KGen}(1^\lambda) \xrightarrow{\$} (sk, pk)$. *On input a security parameter 1^λ, this probabilistic algorithm outputs a secret signing key sk and a public verification key pk.*
- $\mathsf{Sign}(sk, m) \xrightarrow{\$} \sigma$. *On input a signing key sk and a message $m \in \{0,1\}^*$, this (possibly) probabilistic algorithm outputs a signature σ.*
- $\mathsf{Verify}(K, m, \tau) \to \{0,1\}$. *On input a verification key pk, a message m, and a signature σ, this deterministic algorithm outputs 1 (indicating validity of the signature) or 0 (otherwise).*

Definition 2.5 (Existential and strong unforgeability of signatures)**.** *Let* $\mathsf{Sig} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a signature scheme and experiments* $\mathsf{Expt}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{A}}(1^\lambda)$ *and* $\mathsf{Expt}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{A}}(1^\lambda)$ *for an adversary $\mathcal{A}$ be defined as in Figure 2.1.*

We say that Sig *provides* existential (resp. strong) unforgeability under chosen-message attacks *(*EUF-CMA*, resp.* SUF-CMA*) if for all PPT adversaries the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{A}} := \Pr\left[\mathsf{Expt}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{A}}(1^\lambda) = 1\right], \quad \textit{resp.} \quad \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{A}} := \Pr\left[\mathsf{Expt}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{A}}(1^\lambda) = 1\right].$$

2.2.4 Hash Functions and the Random Oracle Model

A (cryptographic) hash function compresses an input message of arbitrary length to a fixed-length hash value. For security we demand that this mapping is collision-resistant, i.e., it should be hard to find two distinct input values that are mapped to the same hash value. In the key exchange protocols we analyze, hash functions are in particular used as underlying building blocks for key derivation and to shorten the representation of communication transcripts. Following their usage in practice, we consider hash functions to be unkeyed and demand that a security reduction to a hash function's collision resistance provides effective means for constructing a concrete algorithm generating a collision (cf. Rogaway [Rog06]).

Definition 2.6 (Hash function and collision resistance)**.** *A* hash function $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$ *maps arbitrary-length messages $m \in \{0,1\}^*$ to a hash value $\mathsf{H}(m) \in \{0,1\}^\lambda$ of fixed length $\lambda \in \mathbb{N}$.*

We say that H *provides* collision resistance *(*COLL*) if we cannot construct an efficient adversary $\mathcal{A}$ for which the following advantage function is non-negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{A}} := \Pr\left[(m, m') \xleftarrow{\$} \mathcal{A} : m \neq m' \textit{ and } \mathsf{H}(m) = \mathsf{H}(m')\right].$$

The random oracle model. In some settings, (practical) constructions for a cryptographic task can only be found when idealizing the security of a certain primitive used. One such idealization is to assume the existence of a truly random function $H\colon \{0,1\}^* \to \{0,1\}^\lambda$ which can be publicly evaluated by every participating party in the system. This idealization was introduced by Bellare and Rogaway [BR93] as the *random oracle model*, referring to H as a *random oracle*. Relying on a random oracle eases the design of cryptographic schemes as follows: for fresh inputs x the output value $H(x)$ can be assumed to be uniformly random; furthermore, in a reduction, the random oracle can be efficiently computed (and potentially programmed on certain values) by sampling output values at random 'on-the-fly' (a technique called 'lazy sampling').

When implementing a cryptographic scheme designed based on a random oracle, the idealized random oracle is instantiated by some concrete implementation of, e.g., a hash function whose output is then assumed to be essentially indistinguishable from that of a random oracle. In general, an actual real-world implementation of a random oracle of course cannot exist and there are counterexamples showing that contrived instantiations can yield insecure schemes [CGH98]. Yet, the random oracle model has turned out to be immensely useful for the design of practical cryptographic schemes and no security weaknesses resulting from this idealization have occurred in such schemes. The random oracle model hence emerged as an essential tool for proving security especially when schemes not relying on a random oracle (in the so-called *standard model*) have not been found (yet) or are too inefficient.

Part I

Key Exchange

Chapter 3

Key Exchange Preliminaries

Summary. In this chapter we recall preliminaries on key exchange and foundational work on the security of key exchange protocols that form the basis of our work in the first part of this thesis. We in particular summarize the seminal work by Bellare and Rogaway [BR94] that formalizes key exchange security for the first time and which underpins essentially all security models for key exchange to date. We furthermore recall cryptographic assumptions that we employ or refer to in our security analyses like, e.g., the Decisional Diffie–Hellman assumption or the pseudorandom-function oracle-Diffie–Hellman assumption, as well as (assumptions on) the HMAC and HKDF functions.

3.1 The Bellare–Rogaway Model

A protocol establishing a secure shared key over an insecure connection between parties sharing only public information has been one of the great inventions in the seminal paper by Diffie and Hellman in 1976 [DH76]. The first rigorous formalization of what security means for such key exchange protocols was then given in the foundational work by Bellare and Rogaway [BR94], formalizing the core security goals of key secrecy and (entity) authentication. Their model considers a strong adversary that controls the network, hence able to execute both passive and active attacks when interacting with multiple executions of the key exchange protocol (spawning them via a NewSession oracle and controlling communication via a Send oracle). The adversary further is allowed to corrupt some of the interacting honest parties (via a Corrupt oracle), learning their long-term secrets, and to reveal the session keys established in some of the protocol runs (through a Reveal oracle). Security then demands that such a powerful adversary is nevertheless unable to distinguish the established session key in an uncompromised (or "fresh") session from a random string (through a Test oracle). This, informally, provides the guarantee that established keys look random to such an adversary and that the (uncompromised) party the key is established with is indeed authenticated. The Bellare–Rogaway model was extended in a number of follow-up works (see also Section 1.3) to capture related protocols and concepts, including the treatment of the public-key setting [BM97, BWJM97], three-party protocols [BR95] and password-based key exchange [BPR00], or forward secrecy (in the sense that keys remain secure even if the adversary later corrupts an involved party's long-term secret [Gün90, DVOW92]) and the compromise of ephemeral secrets [CK01, LLM07].

In the following, we recap the classical notion of key exchange security in the style of the Bellare–Rogaway model for the public-key setting. The formalism used is based on that of Brzuska et al. [BFWW11, Brz13] which we also adopt in the security model we propose for multi-stage key exchange, presented in Chapter 4 based on [FG14, DFGS15a, DFGS16, FG17]. We restrict ourselves here to the case of mutual authentication with pre-specified peers; our

multi-stage model will then also cover settings where one or both parties are unauthenticated and possibly learn the partner identity only during the protocol run.

3.1.1 Notation

We denote by $\mathcal{U}$ the set of *identities* (or *users*) used to model the participants in the system, each identified by some $U \in \mathcal{U}$ and associated with a certified *long-term* public key pk_U and secret key sk_U. We uniquely identity protocol *sessions* (on the administrative level of the model) via a *label* $\mathsf{label} \in \mathsf{LABELS} = \mathcal{U} \times \mathcal{U} \times \mathbb{N}$, where $\mathsf{label} = (U, V, k)$ indicates the k-th local session of identity U (the session *owner*) with V as the intended communication *partner*.

For each session, we maintain the following information in a *session list* $\mathsf{List_S}$, where values in square brackets [] indicate the default, initial value.

- $\mathsf{label} \in \mathsf{LABELS}$: the unique (administrative) session label
- $\mathsf{id} \in \mathcal{U}$: the identity of the session owner
- $\mathsf{pid} \in \mathcal{U}$: the identity of the intended communication partner
- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$: the session owner's role in this session
- $\mathsf{st_{exec}} \in \{\mathsf{running}, \mathsf{accepted}, \mathsf{rejected}\}$: the state of execution, set once to $\mathsf{accepted}$ when the session accepts resp. $\mathsf{rejected}$ when the session rejects [$\mathsf{running}$]
- $\mathsf{sid} \in \{0,1\}^* \cup \{\perp\}$: the session identifier, set once upon acceptance [$\perp$]
- $\mathsf{key} \in \{0,1\}^* \cup \{\perp\}$: the established session key, set once upon acceptance [$\perp$]
- $\mathsf{st_{key}} \in \{\mathsf{fresh}, \mathsf{revealed}\}$: the state of the session key [fresh]
- $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}$: indicator whether the session key key has been tested or not [false]

Adding a not fully specified tuple $(\mathsf{label}, U, V, \mathsf{role})$ to $\mathsf{List_S}$ by convention sets all other entries to their default value. We also use shorthands and write, e.g., $\mathsf{label.sid}$ for the element sid in the tuple with (unique) label label in $\mathsf{List_S}$.

Partnering of sessions. In the key exchange literature there exist different approaches to specify when two sessions are *partnered* in the sense of being considered to have jointly run an execution of the key exchange protocol. The notion of partnering is crucial for excluding trivial attacks in the security notion: An adversary that, in an undisturbed protocol execution between two sessions, tests one session and reveals the (same) session key derived at the communication partner should not be considered successful, as it can trivially check whether the tested key is real or random by comparing it with the revealed key. Forbidding an adversary to test and reveal keys in partnered sessions effectively prevents such trivial attacks.

Here, we follow the approach going back to Bellare, Rogaway, and Pointcheval [BPR00] to define partnering of sessions via session identifiers. Compared to the original partnering definition via matching communication transcripts used by Bellare and Rogaway [BR94], this approach is more versatile and allows, e.g., to omit (cryptographically) unnecessary parts of the transcript. It must in turn be accompanied by a type of soundness condition (captured via the notion of Match security we define later) in order to prevent trivial session identifiers through which more than just the two actual communication partners are considered partnered (e.g., all sessions sharing some fixed session identifier would trivialize security by preventing any session from being revealed).

Formally, we say that two distinct sessions label and label' are *partnered* if both sessions hold the same session identifier, i.e., $\mathsf{label.sid} = \mathsf{label}'.\mathsf{sid} \neq \perp$. For correctness, we require that two sessions jointly executing the key exchange protocol without tampering are partnered upon acceptance.

3.1.2 Adversary Model

We model the adversary against a key exchange protocol as a probabilistic polynomial-time (PPT) Turing machine $\mathcal{A}$ interacting with the protocol via oracles. The adversary is active and controls the full network, i.e., the communication between all parties, enabling interception, injection, and dropping of messages and scheduling their delivery. In order to capture whether certain interactions and conditions are (in-)admissible we set a flag lost (initialized to false) in cases where the adversary trivially loses (such as both revealing and testing the session key in partnered sessions, destroying the freshness of the test session).

The adversary may query the following oracles to interact with the protocol.

- $\mathsf{NewSession}(U, V, \mathsf{role})$: Creates a new session with a (unique) new label label for owner participant identity $\mathsf{id} = U$ with role role, having $\mathsf{pid} = V$ as intended partner. Add (label, U, V, role) to $\mathsf{List_S}$ and return label.

- $\mathsf{Send}(\mathsf{label}, m)$: Sends a message m to the session with label label.

  If there is no tuple with label label in $\mathsf{List_S}$, return $\perp$. Otherwise, run the protocol on behalf of U on message m and return the response and the updated state of execution $\mathsf{label.st_{exec}}$. As a special case, if $\mathsf{label.role} = \mathsf{initiator}$ and $m = \mathsf{init}$, the protocol is initiated (without any input message).

  If the state of execution changes to $\mathsf{label.st_{exec}} = \mathsf{accepted}$ and the intended communication partner pid is corrupted, then set $\mathsf{label.st_{key}} \leftarrow \mathsf{revealed}$.

- $\mathsf{Reveal}(\mathsf{label})$: Reveals the session key $\mathsf{label.key}$ in the session with label label.

  If there is no session with label label in $\mathsf{List_S}$ or $\mathsf{label.st_{exec}} \neq \mathsf{accepted}$, then return $\perp$. Otherwise, set $\mathsf{label.st_{key}}$ to $\mathsf{revealed}$ and provide the adversary with $\mathsf{label.key}$.

- $\mathsf{Corrupt}(U)$: Provide the adversary with the long-term secret sk_U of user U. No further queries are allowed to sessions owned by U.

  In the non-forward-secret case, for each session label owned by U or having U set as intended partner (i.e., $\mathsf{label.id} = U$ or $\mathsf{label.pid} = U$) set $\mathsf{label.st_{key}}$ to $\mathsf{revealed}$. In this case, all (previous and future) session keys are considered to be disclosed.

- $\mathsf{Test}(\mathsf{label})$: Tests the session key of the session with label label. In the security game this oracle is given a uniformly random test bit b_{test} as state which is fixed throughout the game. The adversary may issue this query at most once at an arbitrary point in the game.

  If there is no session with label label in $\mathsf{List_S}$ or if $\mathsf{label.st_{exec}} \neq \mathsf{accepted}$, return $\perp$. Otherwise, set $\mathsf{label.tested}$ to true. If the test bit b_{test} is 0, sample a key $K \xleftarrow{\$} \mathcal{D}$ at random from the session key distribution $\mathcal{D}$. If $b_{\mathsf{test}} = 1$, let $K \leftarrow \mathsf{label.key}$ be the real session key.

  Return K.

3.1.3 Bellare–Rogaway Security

We follow the formalization by Brzuska et al. [BFWW11, Brz13] to separate the security experiments for actual key indistinguishability (BR security) and for session matching (BR-Match

security). The former captures the classical idea of session keys being indistinguishable from random ones and (implicitly) mutually authenticated. The latter formalizes soundness of session identifiers in that they appropriately identify partnered sessions.

BR-Match security. The notion of BR-Match security ensures soundness of the session identifiers sid, i.e., that they properly identify partnered sessions in the sense that

1. sessions with the same session identifier hold the same key,
2. sessions are partnered with the intended (authenticated) participant, and
3. at most two sessions have the same session identifier.

The BR-Match security game $G_{\mathsf{KE},\mathcal{A}}^{\mathsf{BR\text{-}Match}}$ is defined as follows.

Definition 3.1 (BR-Match security)**.** *Let* KE *be a key exchange protocol and* $\mathcal{A}$ *a PPT adversary interacting with* KE *via the queries defined in Section 3.1.2 in the following game* $G_{\mathsf{KE},\mathcal{A}}^{\mathsf{BR\text{-}Match}}$*:*

Setup. *The challenger generates long-term public/private-key pairs for each participant* $U \in \mathcal{U}$*.*

Query. *The adversary* $\mathcal{A}$ *receives the generated public keys and has access to the queries* NewSession*,* Send*,* Reveal*,* Corrupt*, and* Test*.*

Stop. *At some point, the adversary stops with no output.*

We say that $\mathcal{A}$ *wins the game, denoted by* $G_{\mathsf{KE},\mathcal{A}}^{\mathsf{BR\text{-}Match}} = 1$*, if at least one of the following conditions holds:*

1. *There exist two distinct labels* label*,* label$'$ *such that* label.sid $=$ label$'$.sid $\neq \perp$*,* label.$\mathsf{st}_{\mathsf{exec}} \neq$ rejected*, and* label$'$.$\mathsf{st}_{\mathsf{exec}} \neq$ rejected*, but* label.key $\neq$ label$'$.key*. (Different session keys in partnered sessions.)*
2. *There exist two distinct labels* label*,* label$'$ *such that* label.sid $=$ label$'$.sid $\neq \perp$*,* label.role $=$ initiator*, and* label$'$.role $=$ responder*, but* label.pid $\neq$ label$'$.id *or* label.id $\neq$ label$'$.pid*. (Different intended partner.)*
3. *There exist three pairwise distinct labels* label*,* label$'$*,* label$''$ *such that* label.sid $=$ label$'$.sid $=$ label$''$.sid $\neq \perp$*. (More than two sessions share the same session identifier.)*

We say KE *is* BR-Match-secure *if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}_{\mathsf{KE},\mathcal{A}}^{\mathsf{BR\text{-}Match}} := \Pr\left[G_{\mathsf{KE},\mathcal{A}}^{\mathsf{BR\text{-}Match}} = 1\right].$$

BR security. The second and actual key secrecy notion, BR security, is defined as follows.

Definition 3.2 (BR security)**.** *Let* KE *be a key exchange protocol with key distribution* $\mathcal{D}$ *and* $\mathcal{A}$ *a PPT adversary interacting with* KE *via the queries defined in Section 3.1.2 in the following game* $G_{\mathsf{KE},\mathcal{A}}^{\mathsf{BR},\mathcal{D}}$*:*

Setup. *The challenger chooses the test bit* $b_{\mathsf{test}} \xleftarrow{\$} \{0,1\}$ *at random and sets* lost $\leftarrow$ false*. It furthermore generates long-term public/private-key pairs for each participant* $U \in \mathcal{U}$*.*

Query. *The adversary* $\mathcal{A}$ *receives the generated public keys and has access to the queries* NewSession*,* Send*,* Reveal*,* Corrupt*, and* Test*. Note that such queries may set* lost *to* true*.*

Guess. *At some point, $\mathcal{A}$ stops and outputs a guess b.*

Finalize. *The challenger sets the 'lost' flag to* $\mathsf{lost} \leftarrow \mathsf{true}$ *if there exist two (not necessarily distinct) labels* label, label' *such that* $\mathsf{label.sid} = \mathsf{label}'.\mathsf{sid}$, $\mathsf{label.st_{key}} = \mathsf{revealed}$, *and* $\mathsf{label}'.\mathsf{tested} = \mathsf{true}$. *(Adversary has tested and revealed the key in a single session or in two partnered sessions.)*

We say that $\mathcal{A}$ wins the game, denoted by $G^{\mathsf{BR},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} = 1$, if $b = b_{\mathsf{test}}$ and $\mathsf{lost} = \mathsf{false}$. *Note that the winning condition is independent of the forward secrecy property of* KE, *as forward secrecy is already reflected in the* Corrupt *query.*

We say KE *is* BR-secure *in a non-forward-secret resp. forward-secret manner if* KE *is* BR-Match*-secure and for all PPT adversaries $\mathcal{A}$ the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{BR},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{BR},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} = 1\right] - \frac{1}{2}.$$

3.2 Cryptographic Assumptions for Key Exchange

In the following we recall established cryptographic assumptions employed in analyses of key exchange protocols as well as (variants of) the more recent pseudorandom-function oracle-Diffie–Hellman (PRF-ODH) assumption and specific assumptions on the HMAC-based key derivation function HKDF.

3.2.1 The Diffie–Hellman Assumptions

Along with their seminal proposal of a key exchange protocol, Diffie and Hellman [DH76] introduced the assumption that, in a finite cyclic group $\mathbb{G}$ of prime order q with generator g it should be hard to compute g^{ab} given g^a and g^b. (Here, $\mathbb{G}$ is an instance from a sequence of groups $(\mathbb{G}_\lambda)_\lambda$ in dependency of the security parameter λ.) This assumption is by now well-known as the computational Diffie–Hellman (CDH) assumption.

Definition 3.3 (Computational Diffie–Hellman (CDH) assumption)**.** *Let $\mathbb{G}$ be a cyclic group of prime order q from a sequence of groups $(\mathbb{G}_\lambda)_\lambda$ in dependency of the security parameter λ, and g be a generator of $\mathbb{G}$. The* computational Diffie–Hellman (CDH) assumption *states that for any probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$ the following advantage function is negligible in λ:*

$$\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G},\mathcal{A}} := \Pr\left[\mathcal{A}(1^\lambda, \mathbb{G}, g, g^a, g^b) = g^{ab} \;\middle|\; a, b \xleftarrow{\$} \mathbb{Z}_q\right].$$

For key exchange security in the sense of Bellare–Rogaway we require that session keys be indistinguishable from random. The CDH assumption is too weak for that purpose, as an adversary while not able to recover the full session key (e.g., g^{ab}) may still learn (biases) of certain bits of the key. In key exchange, one hence rather employs the (conjectured to be stronger) *decisional* Diffie–Hellman assumption [Bon98] demanding that Diffie–Hellman values be indistinguishable from randomly chosen group elements.

Definition 3.4 (Decisional Diffie–Hellman (DDH) assumption)**.** *Let $\mathbb{G}$ be a cyclic group of prime order q with generator g from a sequence of groups $(\mathbb{G}_\lambda)_\lambda$ in dependency of the security parameter λ. The* decisional Diffie–Hellman (DDH) assumption *states that for any probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$ the following advantage function is negligible in λ:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\mathcal{A}} := \Big|\, &\Pr\left[\mathcal{A}(1^\lambda, \mathbb{G}, g, g^a, g^b, g^{ab}) = 1 \;\middle|\; a, b \xleftarrow{\$} \mathbb{Z}_q\right] \\ &- \Pr\left[\mathcal{A}(1^\lambda, \mathbb{G}, g, g^a, g^b, g^c) = 1 \;\middle|\; a, b, c \xleftarrow{\$} \mathbb{Z}_q\right] \Big|.\end{aligned}$$

A common strategy for security proofs of Diffie–Hellman-based key exchange protocol is to employ the random oracle model (cf. Section 2.2.4) and reduce security down to being able to solve the computational Diffie–Hellman problem while having access to a decisional Diffie–Hellman problem. This problem is known as the *Gap*-Diffie–Hellman (GapDH) assumption [OP01] and enables proofs for example in settings where Diffie–Hellman shares are re-used across multiple protocol instances. The GapDH has been used in analyses of many Diffie–Hellman-based key exchange protocol (e.g., [JP02, KP05, LM06, DF11]) and we will employ it in our analysis of the QUIC protocol (see Chapter 5).

Definition 3.5 (Gap-Diffie–Hellman (GapDH) assumption)**.** *Let $\mathbb{G}$ be a cyclic group of prime order q with generator g from a sequence of groups $(\mathbb{G}_\lambda)_\lambda$ in dependency of the security parameter λ. Let* $\mathrm{DDH}(\cdot,\cdot,\cdot)$ *be a decisional Diffie–Hellman (*DDH*) oracle that checks for Diffie–Hellman tuples in the sense that* $\mathrm{DDH}(X,Y,Z) = 1$ *if and only if* $\log_g Z = \log_g X \cdot \log_g Y \mod q$*. The* Gap-Diffie–Hellman (GapDH) assumption *states that for any probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$ the following advantage function is negligible in λ:*

$$\mathsf{Adv}^{\mathsf{GapDH}}_{\mathbb{G},\mathcal{A}} := \Pr\left[\mathcal{A}^{\mathrm{DDH}(\cdot,\cdot,\cdot)}(1^\lambda, \mathbb{G}, g, g^a, g^b) = g^{ab} \;\middle|\; a, b \xleftarrow{\$} \mathbb{Z}_q\right].$$

A variant of GapDH where the first input to the DDH oracle $\mathrm{DDH}(g^a,\cdot,\cdot)$ is fixed to g^a is known as the *Strong* Diffie–Hellman (StDH) assumption.

3.2.2 The PRF-ODH Assumption(s)

The pseudorandom-function oracle-Diffie–Hellman (PRF-ODH) assumption has been introduced by Jager et al. [JKSS12] in their analysis of the TLS 1.2 key exchange. It is a variant of the oracle-Diffie–Hellman assumption introduced by Abdalla et al. [ABR01] in the context of the DHIES encryption scheme. Basically, the PRF-ODH states that the pseudorandom-function value $\mathsf{PRF}(g^{uv}, x^\star)$ for a Diffie–Hellman-type key g^{uv} is indistinguishable from a random string, even when given g^u and g^v and when being able to see related values $\mathsf{PRF}(S^u, x)$ and/or $\mathsf{PRF}(T^v, x)$ for chosen values S, T, and x.

As we will see in our analyses of various TLS 1.3 handshake modes, the PRF-ODH assumption appears quite naturally in Diffie–Hellman-based key exchange protocols, e.g., when aiming at strong security of unauthenticated session keys. More specifically, an adversary may have a tested session accept with a session key $\mathsf{PRF}(g^{uv}, \dots)$ derived from g^{uv} for some honest Diffie–Hellman shares g^u and g^v while making the honest participant that sent g^u accept with an adversary-controlled share $g^{v'}$, deriving a key $\mathsf{PRF}(g^{uv'}, \dots)$ from $g^{uv'}$. In a Bellare–Rogaway-style security game, a reduction encoding a (Decisional) Diffie–Hellman challenge in g^u and g^v now needs to be to respond to a Reveal query on $\mathsf{PRF}(g^{uv'}, \dots)$, while knowing neither u nor v'. This is where the PRF-ODH assumption enables a sound simulation by providing access to the related session key value $\mathsf{PRF}(g^{uv'}, \dots)$.

Different variants of the PRF-ODH assumption have been introduced and used in the literature for the analysis of various key exchange protocols. The original version by Jager et al. [JKSS12] for the ephemeral Diffie–Hellman TLS 1.2 handshake required security under a single query for one of the two Diffie–Hellman shares. Krawczyk et al. [KPW13] extended this to multiple oracle queries against this share for their analyses of the TLS 1.2 static Diffie–Hellman handshake. In our work on the TLS 1.3 Diffie–Hellman-based 0-RTT handshake (see Chapter 7) we augmented the PRF-ODH assumption by an additional single oracle query to the other Diffie–Hellman share. Finally, Brendel and Fischlin [BF17] required multiple queries to both shares in their work on a 0-RTT extension to the Extended Access Control (EAC) protocol.

The PRF-ODH assumption can be seen as a more modular building block for proofs compared to the rather involved proof strategies based on the GapDH or StDH assumption in the random

oracle model. We refer to our recent study [BFGJ17] for instantiations of PRF-ODH under different assumptions, but note that all variants can be instantiated in the random oracle model under the StDH assumption. Our study furthermore establishes relations between the different variants and provides indication that PRF-ODH is likely not a standard-model assumption.

In the following, we present the generic PRF-ODH assumption definition we put forward in [BFGJ17] which captures all different flavors occurring in previous works. This in particular includes the original Jager et al. [JKSS12] single-one-sided (snPRF-ODH) variant we employ in all our key secrecy analyses of Diffie–Hellman-based TLS 1.3 handshakes (see Chapters 6 and 7) as well as the (multiple-single) double-sided (msPRF-ODH) variant we rely on for our analysis of the TLS 1.3 Diffie–Hellman 0-RTT handshake (in Chapter 7).

Definition 3.6 (Generic PRF-ODH assumption)**.** *Let $\lambda \in \mathbb{N}$ be the security parameter and $\mathbb{G}$ be a cyclic group of prime order q with generator g. Let $\mathsf{PRF}\colon \mathbb{G} \times \{0,1\}^* \to \{0,1\}^\lambda$ be a pseudorandom function that takes a key $K \in \mathbb{G}$ and a label $x \in \{0,1\}^*$ as input and outputs a value $y \in \{0,1\}^\lambda$, i.e., $y \leftarrow \mathsf{PRF}(K, x)$.*

We define a generic security notion lrPRF-ODH *which is parameterized by $\mathsf{l}, \mathsf{r} \in \{\mathsf{n}, \mathsf{s}, \mathsf{m}\}$ indicating how often the adversary is allowed to query a certain "left", resp. "right", oracle (ODH_u, resp. ODH_v) where n indicates that no query is allowed, s that a single query is allowed, and m that multiple (polynomially many) queries are allowed to the respective side. Consider the following security game $G^{\mathsf{lrPRF\text{-}ODH}}_{\mathsf{PRF},\mathbb{G},\mathcal{A}}$ between a challenger and a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$.*

1. *The challenger samples $u \xleftarrow{\$} \mathbb{Z}_q$ and provides $\mathbb{G}$, g, and g^u to the adversary $\mathcal{A}$.*
2. *If $\mathsf{l} = \mathsf{m}$, $\mathcal{A}$ can issue arbitrarily many queries to the following oracle ODH_u.*

   ODH_u **oracle.** *On a query of the form (S, x), the challenger first checks if $S \notin \mathbb{G}$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{PRF}(S^u, x)$ and returns y.*
3. *Eventually, $\mathcal{A}$ issues a challenge query $x^\star$. Upon this query, the challenger samples $v \xleftarrow{\$} \mathbb{Z}_q$ and a bit $b \xleftarrow{\$} \{0,1\}$ uniformly at random. It then computes $y_0^\star = \mathsf{PRF}(g^{uv}, x^\star)$ and samples $y_1^\star \xleftarrow{\$} \{0,1\}^\lambda$ uniformly random. The challenger returns $(g^v, y_b^\star)$ to $\mathcal{A}$.*
4. *Next, $\mathcal{A}$ may issue (arbitrarily interleaved) queries to the following oracles ODH_u and ODH_v (depending on l and r).*

   ODH_u **oracle.** *The adversary $\mathcal{A}$ may ask no ($\mathsf{l} = \mathsf{n}$), a single ($\mathsf{l} = \mathsf{s}$), or arbitrarily many ($\mathsf{l} = \mathsf{m}$) queries to this oracle. On a query of the form (S, x), the challenger first checks if $S \notin \mathbb{G}$ or $(S, x) = (g^v, x^\star)$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{PRF}(S^u, x)$ and returns y.*

   ODH_v **oracle.** *The adversary $\mathcal{A}$ may ask no ($\mathsf{r} = \mathsf{n}$), a single ($\mathsf{r} = \mathsf{s}$), or arbitrarily many ($\mathsf{r} = \mathsf{m}$) queries to this oracle. On a query of the form (T, x), the challenger first checks if $T \notin \mathbb{G}$ or $(T, x) = (g^u, x^\star)$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{PRF}(T^v, x)$ and returns y.*
5. *At some point, $\mathcal{A}$ stops and outputs a guess $b' \in \{0,1\}$.*

We say that the adversary wins the lrPRF-ODH *game if $b' = b$ and define the advantage function as*

$$\mathsf{Adv}^{\mathsf{lrPRF\text{-}ODH}}_{\mathsf{PRF},\mathbb{G},\mathcal{A}} := 2 \cdot \left(\Pr[b' = b] - \frac{1}{2} \right).$$

Assuming a sequence of groups in dependency of the security parameter, we say that a pseudorandom function PRF *with keys from $(\mathbb{G}_\lambda)_\lambda$ provides* lrPRF-ODH *security (for $\mathsf{l}, \mathsf{r} \in \{\mathsf{n}, \mathsf{s}, \mathsf{m}\}$) if for any $\mathcal{A}$ the advantage $\mathsf{Adv}^{\mathsf{lrPRF\text{-}ODH}}_{\mathsf{PRF},\mathbb{G},\mathcal{A}}$ is negligible in the security parameter λ.*

3.2.3 Assumptions on HMAC and HKDF

Key exchange protocols rely on key derivation functions in order to extract uniformly random session keys of a certain length from potentially non-uniform sources of entropy (e.g., Diffie–Hellman group elements). The key exchanges we analyze in the QUIC protocol and TLS 1.3 protocol drafts employ HKDF [Kra10, KE10], a specific hash-function-based key derivation function employing HMAC [BCK96, KBC97] as core building block. We here recap the definitions for both. For our analyses we rely on several assumptions on both functions, including standard PRF security (cf. Definition 2.2.1), the PRF-ODH assumption, and the random oracle model, as well as two more specific assumptions given below.

HMAC [BCK96, KBC97] is a MAC scheme based on a cryptographic hash function H. Key generation simply samples a random key $K \xleftarrow{\$} \{0,1\}^\lambda$, tagging and verification is then done by (re)computing on a message m the MAC value $\mathsf{HMAC}(K,m) := \mathsf{H}((K \oplus \mathsf{opad}) \,\|\, \mathsf{H}((K \oplus \mathsf{ipad}) \,\|\, m))$, where opad and ipad are two one-block-long padding values consisting of repeated bytes `0x5c` and `0x36`, respectively.

HKDF is a key derivation function following the *extract-then-expand* paradigm [Kra10, KE10] instantiated with HMAC. We adopt the standard notation for the two HKDF functions: $\mathsf{HKDF.Extract}(XTS, SKM)$ on input an (non-secret and potentially fixed) extractor salt XTS and some (not necessarily uniform) source key material SKM outputs a pseudorandom key PRK. $\mathsf{HKDF.Expand}(PRK, CTXinfo)$ on input a pseudorandom key PRK (from the Extract step) and some (potentially empty) context information $CTXinfo$ outputs pseudorandom key material KM.[3] Both functions are instantiated with HMAC, where directly $\mathsf{HKDF.Extract}(XTS, SKM) := \mathsf{HMAC}(XTS, SKM)$ and HKDF.Expand iteratively invokes HMAC to generate pseudorandom output of the required length (see [Kra10]).

The first assumption, which we denote as $\mathsf{HMAC}(0,\$)$-$\$$, concerns HMAC and is induced by an HKDF extraction step of a uniformly random pre-shared key in the TLS 1.3 `draft-14` key schedule (see Section 7.3). It states that $\mathsf{HMAC}(0,x)$ for an unknown, uniformly random value $x \xleftarrow{\$} \{0,1\}^\lambda$ is computationally indistinguishable from another uniformly random value $y \xleftarrow{\$} \{0,1\}^\lambda$.

Definition 3.7 ($\mathsf{HMAC}(0,\$)$-$\$$ assumption)**.** *Let* HMAC *be the HMAC function as defined in [BCK96]. We say that the* $\mathsf{HMAC}(0,\$)$-$\$$ *assumption holds for* HMAC *if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{HMAC}(0,\$)\text{-}\$}_{\mathsf{HMAC},\mathcal{A}} := \Big|\Pr\left[\mathcal{A}(1^\lambda, \mathsf{HMAC}(0,x)) = 1 \;\middle|\; x \xleftarrow{\$} \{0,1\}^\lambda\right] - \Pr\left[\mathcal{A}(1^\lambda, y) = 1 \;\middle|\; y \xleftarrow{\$} \{0,1\}^\lambda\right]\Big|.$$

The second assumption, induced by the TLS 1.3 `draft-12` key schedule (see Section 7.5), is that the HKDF extraction step HKDF.Extract (being the HMAC function) is a strong (computational) extractor (in a mild form, extracting λ uniform bits from λ uniform bits with a public seed of λ uniform bits).

Definition 3.8 (HKDF.Extract as strong (computational) extractor)**.** *Let* $\mathsf{HKDF.Extract}\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ *be the HKDF extraction function as defined in [Kra10] (i.e.,* $\mathsf{HKDF.Extract} = \mathsf{HMAC}$*) with fixed inputs and output length* λ*. We say that* HKDF.Extract *is a strong (computational) extractor (in that setting) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{st\text{-}Extract}}_{\mathsf{HKDF.Extract},\mathcal{A}} := \Big|\Pr\left[\mathcal{A}(1^\lambda, x, \mathsf{HKDF.Extract}(x,y)) = 1 \;\middle|\; x,y \xleftarrow{\$} \{0,1\}^\lambda\right] - \Pr\left[\mathcal{A}(1^\lambda, x, y) = 1 \;\middle|\; x,y \xleftarrow{\$} \{0,1\}^\lambda\right]\Big|.$$

[3] For simplicity, we omit the original third parameter L in Expand determining its output length and always assume that $L = \lambda$ for our security parameter λ if not stated otherwise.

Chapter 4

Multi-Stage Key Exchange

Summary. In this chapter we introduce our security model for multi-stage key exchange protocols, enabling our analyses of the QUIC (Chapter 5) and TLS 1.3 (Chapters 6 and 7) protocols. We begin by briefly introducing the setting of multi-stage key exchange protocols and why existing key exchange security models are insufficient to capture such protocols. We then provide an overview over the features and advanced security aspects captured by our model, before defining the technical model itself and its associated security notions. Afterwards, we discuss under which conditions the keys established in a multi-stage key exchange protocol can be securely composed with a subsequent symmetric-key protocol and provide an according composition theorem. At the end, we briefly discuss other applications our model has found since its introduction. The multi-stage key exchange model was introduced in a work published at ACM CCS 2014 [FG14] and extended in works published at ACM CCS 2015 [DFGS15a] and IEEE EuroS&P 2017 [FG17], and presented at the TLS 1.3 TRON workshop 2016 [DFGS16]. The version presented in this chapter is based on the most comprehensive one from [FG17].

4.1 Introduction

The goal of an authenticated key exchange protocol is to establish a cryptographically secure key between two parties over an insecure network, enabling the two parties securely communicate using the key established. Traditionally, the key exchange step was seen as the setup phase for the actual communication, ceasing as soon as the key is established, and this key being the only information passed to the following protocol flow (cf. Chapter 3).

Secure communication protocols in practice, however, demand a more flexible use of key exchange protocols where multiple keys are established in a continuous process, interleaving derivation of keys with their usage. Two prominent examples are the QUIC protocol [QUI] proposed by Google in 2013 and the upcoming next version of the Transport Layer Security (TLS) protocol, TLS 1.3 [Res18], designed by the Internet Engineering Task Force (IETF), whose key exchange designs and security we study in Chapters 5–8 of this thesis.

QUIC. Google proposed QUIC (for "Quick UDP Internet Connections") as a low-latency transport protocol with security equivalent to TLS. It is a Diffie–Hellman-based connection establishment protocol, with the particular goal of reduced round complexity. Specifically, it aims at zero round-trip time (0-RTT) key establishment, i.e., the client is enabled to immediately deliver data along with the first message sent to the server, protected under an intermediate cryptographic key. With the server's reply and contribution to the key exchange, both parties then switch to a stronger key and continue the communication with that key.

TLS 1.3. TLS 1.3 is the next version of the Transport Layer Security (TLS) protocol currently drafted by the IETF. The protocol design includes several substantial cryptographic changes compared to the previous TLS version 1.2 [DR08]. Most notably from the structural perspective of the key exchange design are the following three features of the TLS 1.3 key exchange (the so-called handshake). First, it derives an intermediate key to encrypt some of its key exchange messages for increased privacy. Second, besides the main key for securing application data it establishes further keys for session resumption and exporting of key material. Third, it provides an optional low-latency (0-RTT) mode in which an additional key is derived early on, enabling the client to immediately send data.

4.1.1 Multi-Stage Key Exchange

Both examples, QUIC and TLS 1.3, demonstrate that traditional key exchange models are inappropriate to capture recent desirable key exchange designs in practice. Only considering a single key established, they cannot capture scenarios in which multiple keys are derived, possibly used in between (and within the channel protocol), and being dependent on each other's security. Our goal here is hence to define a security model for such *multi-stage key exchange protocols* which captures the security of all derived keys (at potentially varying security levels) and their dependency in a single framework.

Our starting point will be the Bellare–Rogaway key exchange model (cf. Section 3.1), as it is liberal enough to capture many protocols, but also provides reasonably strong security guarantees. We prudently use the formalization in [Brz13, BFWW11], as we can then more easily argue about composability with arbitrary symmetric-key protocols. A major difference with the single-stage case lies in the dependencies of the different stages. In QUIC, for instance, the final key is protected under the stage-one key by sending the server's ephemeral Diffie–Hellman key through a secure channel. This example indicates that we need to carefully devise and motivate when session keys should be considered fresh (and thus indistinguishable from random) in the sense that they are not trivially available to the adversary.

Our model furthermore allows us to consider unilaterally and mutually authenticated as well as as anonymous sessions of a key exchange protocol. This specifically captures that, e.g., in TLS 1.3, client and server may decide adaptively whether or not to authenticate in a particular execution of the protocol. Authentication in our model can be based either on public keys (as in QUIC or the Diffie–Hellman-based TLS 1.3 handshake) or on pre-shared secrets (as in the TLS 1.3 resumption handshake).

Finally, our model is able to capture low-latency 0-RTT key establishment. Here, the multi-stage treatment allows us to cover security dependencies between the 0-RTT and main communication keys established (as in QUIC). We furthermore formally capture in our model the reduced security properties arising from potential replays of 0-RTT keys possible, e.g., in the TLS 1.3 0-RTT mode design.

The model we present here evolved over several research papers studying the security of QUIC [FG14] and TLS 1.3 in several drafts and handshake modes [DFGS15a, DFGS16, FG17]. In the following, we present the model in its most generic variant described in [FG17], enabling us to capture all aspects of QUIC and TLS 1.3 in the following chapters.

4.2 Overview

Before diving into the technical details of the multi-stage key exchange model, let us provide an overview and some motivation of the changes originating from the multi-stage setting.

Our model for multi-stage key exchange protocols follows the paradigm of Bellare and Rogaway [BR94] in the formalism by Brzuska et al. [BFWW11, Brz13] (cf. Section 3.1). We

likewise keep lists of session information, including values $\mathsf{st}_{\mathsf{exec}}$ about the state (accepted, running, or rejected), the session key key, the status $\mathsf{st}_{\mathsf{key}}$ of the key (fresh or revealed), and a session identifier sid. In these lists, we basically account for the multi-stage setting by storing vectors of these core entries and a variable stage describing the stage a session is in.

Adversarial interaction. As in the basic setting, the adversary can interact with sessions via core oracle queries $\mathsf{NewSession}$, Send, Reveal, $\mathsf{Corrupt}$, and Test in order to initiate a new session, send messages to that session, reveal the session key, corrupt the long-term secret key of a party, and test a session key against a random key, respectively.

One difference in the oracle behavior of our model is owed to the fact that an execution can continue after some session has accepted and derived an intermediate key which can potentially be tested. To enable the adversary to decide whether to test such keys prior to their usage we pause the reply to such Send command after acceptance.

We furthermore distinguish between derived keys which are used *internally* within the key exchange protocol (e.g., to encrypt part of the communication as in QUIC or TLS 1.3) and those only used *externally* (e.g., to encrypt application data). In case of testing an internal session key and returning the genuine or a random key to the adversary, we let the subsequent key exchange step—which may now depend on this session key in the multi-stage setting—also use the genuine or the random key. Otherwise, distinguishing the session keys from random might be trivial.

In contrast to the basic Bellare–Rogaway setting and most other key exchange models, we moreover allow the adversary to issue not only a single but multiple Test queries. This intuitively captures a setting where an adversary concurrently challenges the security of multiple keys derived, e.g., in various stages of a single session. It turns out that both variants are not always immediately reducible via a straightforward hybrid argument, but may depend on some technical conditions like session matching being publicly decidable (cf. similar conditions for Bellare–Rogaway composition [BFWW11, Brz13]) or computational independence of the session keys (see below). For the QUIC and TLS 1.3 draft key exchange protocols we consider in this thesis we will establish such hybrid arguments within the proofs and prefer to make them explicit through considering multiple Test queries.

Forward secrecy. We make the usual distinction between non-forward secrecy and forward secrecy, where the latter protects sessions that accepted before corruption took place. In our multi-stage setting, session keys may however become forward-secret starting only from a certain stage on (e.g., in QUIC only the second key is forward-secret). We hence introduce the notion of *stage-j forward secrecy*, indicating that keys from stage j on are forward-secret.

Key independence. We also differentiate between (session-)key-dependent and (session-)key-independent multi-stage protocols. The difference is basically that, for key-dependent schemes, the session key of stage i is used to derive the session key of stage $i+1$, typically to enhance the security properties of the session keys. QUIC is an example of such a protocol, where the second key exchange phase is run through a channel secured with the first key, making the authenticity of the former dependent on the latter. This property directly affects the adversary's capabilities in the sense that we cannot allow the adversary to reveal the session key of stage i before the key of stage $i+1$ is established in order to avoid trivial attacks. For key-independent protocols, exposure of the preceding session key in contrast does not weaken the next session key (e.g., TLS 1.3 is key-independent, as keys are derived computationally independent of each other).

Public, pre-shared, and semi-static keys. Traditional key exchange models consider long-term keys to be either public keys (e.g., backed by a public-key infrastructure) or pre-shared symmetric keys. We likewise provide two variants of the multi-stage key exchange model, the regular one (MSKE) for the more common public-key scenario as well as a pre-shared secret (MS-PSKE) variant. In the former, public keys are considered authentically distributed upfront, while in the latter, the adversary is given a NewSecret oracle to generate and register new secrets shared between two parties.

In the public-key variant of the model, motivated both by QUIC and intermediate TLS 1.3 drafts, we additionally consider temporary or *semi-static* keys. These keys are somewhat in between ephemeral keys and static keys. For example, QUIC suggests to let the server use the short-term key in the second stage in multiple sessions. The description [LC16] speaks of a life span of about 60 seconds in which the same key is used in every session of this server. Hence, semi-static keys, analogous to static keys, are not bound to a single session. At the same time, they may be too transient to be susceptible to cryptanalytic attacks, such that we do not reveal these key in case of a Corrupt query, but instead introduce a specific RevealSemiStaticKey query. In the model, to avoid introduction of timing events, we let the adversary decide when parties should switch to a new semi-static key via a NewSemiStaticKey command.

Note that the NewSemiStaticKey and RevealSemiStaticKey queries can be omitted for analyses of protocols that do not comprise temporary keys *without* affecting the remaining security model.

Authentication. Our model captures all three common authentication types of key exchange protocols: unauthenticated key exchange where neither party authenticates, unilateral authentication in which only the server (responder) side authenticates, and mutual authentication. As the authentication type might differ between the different stages of a multi-stage key exchange protocol (e.g., depending on the sub-protocol of TLS 1.3 used), the model allows to determine the authentication level for each stage individually. We capture this by allowing the adversary in the security model to determine the type of authentication, and thus the corresponding sub-protocol, when initializing a session. We also allow executions of different types to run *concurrently*, even within a single party.

We additionally allow the communication partner of a session to be unknown at the start of the protocol, i.e., we allow for "post-specified peers" as introduced by Canetti and Krawczyk [CK02a]. In our model, this is captured by letting the adversary initialize a session with a wildcard '$*$' as the intended communication partner. This corresponds to the regular case, e.g., in TLS 1.3 that parties discover their peer's identity during protocol execution when they receive their peer's certificate.

Finally, we aim at strong key secrecy properties for sessions communicating with unauthenticated partners even in cases of incomplete communication. Since the adversary can easily impersonate the unauthenticated party and thereby legitimately compute the shared session key, we cannot in general allow all sessions with unauthenticated partners to be tested. However, if an *honest* unauthenticated partner contributed the cryptographic material needed for the shared session key, then the key between these honest parties should still be secure (and a Test allowed), even if the adversary did not deliver all messages to that honest partner session. To capture this, we introduce the notion of *contributive identifiers*, identifying sessions of honest parties which are currently not partnered according to (full) session identifiers, but indicating that the key is entirely based on an honest peer's contribution. For soundness we assume that partnered sessions (having matching session identifiers) also agree on the contributive identifier. Both session identifiers and contributive identifiers are set primarily as administrative tokens by the key exchange protocol during the execution. In contrast to session identifiers, a contributive identifier can be updated several times instead of being set only once, e.g., to eventually match

the session identifier.[4]

Replayability. The standard approach in key exchange protocols to prevent a man-in-the-middle attacker from replaying messages in order to make a party derive the same key twice is to include a nonce ('number used once') in both the client's and the server's messages and let the nonce contribute to the derived key. In a 0-RTT key exchange, which is essentially a one-pass (i.e., one-message) key exchange protocol [BWM99a], messages (and hence keys) are however—at first glance—inevitably replayable[5].

The QUIC protocol side-stepped the replay problem in its original cryptographic design of Revision 20130620 [LC13] by demanding the server to store all client nonces seen in a so-called "strike register"—restricted in size by a server-specific "orbit" prefix and current time contained in the nonces—and rejecting any recurring nonce. TLS 1.3, in contrast and in response to generic application-level replay attacks on 0-RTT key exchange that we discuss in Chapter 7, forgoes any cryptographically strong protection mechanisms and instead accepts replays as inevitable (on the channel level).

In our model we therefore distinguish between replayable and non-replayable stages (and, hence, keys). The latter type of stage leaves the original security properties untouched. The replayable kind of stage allows for multiple collisions among session identifiers and keys, such that we need to relax the notion of Match security for such stages. Key secrecy in contrast should be not affected by replayability because the adversary may be able to impose the same key on multiple sessions, but the key itself should still look random. For the case of TLS 1.3, this allows us to capture that the 0-RTT key exchange messages of a client session can be replayed to multiple server sessions which will all derive the same key.

Furthermore, we capture the effects of exposures of semi-static keys (via RevealSemiStaticKey) used to non-interactively establish keys in a Diffie–Hellman-based 0-RTT key exchange on both 0-RTT keys (which will be compromised) and non–0-RTT keys (which are required to remain secure).

Secret compromise paradigm. Finally, let us summarize and exemplify the types of secrecy compromise our model considers, and which kind of compromises are beyond the scope of this model.

Our model captures the leakage of long-term secret keys and output session keys as well as the leakage of semi-static keys (in the public-key variant), following the paradigm of Bellare and Rogaway [BR94] to capture the compromise of long(er)-lived secret inputs and key outputs of a key exchange protocol. Note that forward secrecy refers (only) to disclosure of long-term secrets: if, in the public-key variant, semi-static keys are revealed we in any case expect keys to look random, except for the ones in replayable stages. Vice versa, we expect that the compromise of long-term secrets alone (i.e., without also exposing the semi-static key involved) does not affect keys in (forward-secret) replayable stages.

We note that other classical key exchange models by Canetti and Krawczyk [CK01] and LaMacchia et al. [LLM07] further capture the leakage of internal session state or ephemeral secret inputs, which we do not, and against most of which QUIC and TLS 1.3 also do not aim to protect. One can in principal augment our model with such queries, though.

[4] Contributive identifiers may be seen as the identifier-based analogue to prefix-matching definitions used in ACCE models [JKSS12], allowing the adversary to issue Test queries to sessions that are non-trivial to break but normally force the adversary to lose the game. A similar concept was also introduced by Cremers and Feltz [CF12] under the term "origin-sessions" for partnering based on matching conversations, and by Bhargavan et al. [BFK+14] using "(peer-)exchange variables" as alternative to session identifier–based partnering.

[5] We use the notion of replays interchangeably for both messages and the keys computed based on those replayed messages.

At the examples of QUIC (cf. Chapter 5) and TLS 1.3 (cf. Chapters 6–7), this means we consider the leakage of:

- *Long-term keys* (e.g., the server's static Diffie–Hellman key in QUIC or signing resp. pre-shared key in TLS 1.3).
  This is allowed since usage of keys over a long time period induces a substantial risk of compromise, spawning the notion of forward secrecy. Leakage of long-term keys is modeled by the Corrupt query.

- *Session keys* (e.g., the various traffic/channel encryption keys established in QUIC and TLS 1.3 or the derived resumption secret in TLS 1.3).
  This is allowed since session keys are the actual output of a key exchange, used in a follow-up protocol (e.g., for encryption, resumption, or key export) or internally in a subsequent key exchange step (treated in our notion of key independence). Leakage of session keys is modeled by the Reveal query.

- *Semi-static keys* (i.e., medium-lived Diffie–Hellman exponents in QUIC or TLS 1.3's Diffie–Hellman-based 0-RTT mode).
  This is allowed since these keys are meant to be used (by servers) in several connections with multiple clients and over a potentially significant time span. Furthermore, their leakage affect the security of derived replayable keys. Leakage of semi-static keys is modeled by the RevealSemiStaticKey query.

We do not permit the leakage of:

- *Ephemeral secrets* (e.g., the randomness in a signature algorithm or ephemeral Diffie–Hellman exponents).
  This is disallowed as QUIC and TLS 1.3 do not aim (neither achieve) being secure against compromises of this kind.[6]

- *Internal values / session state* (e.g., internally computed master secrets or MAC keys).
  This is disallowed as, again, QUIC and TLS 1.3 do not provide protection against such leakage.

4.3 Preliminaries

In our model, we explicitly separate some *protocol-specific* properties (as, e.g., various authentication flavors) from *session-specific* properties (as, e.g., the state of a running session). We represent protocol-specific properties as a vector $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ that captures the following:

- $\mathsf{M} \in \mathbb{N}$: the number of stages (i.e., the number of keys derived)[7]

[6] To be precise, the OPTLS key exchange protocol underlying the TLS 1.3 handshake actually maintains secrecy of the application traffic key even under exposure of the *server's* ephemeral Diffie–Hellman exponent, given that the client's exponent as well as the server's semi-static key exponent remain secret, as analyzed by Krawczyk and Wee [KW15, KW16]. We omitted capturing one-sided ephemeral secret leakage in our model in order to not further increase its complexity, but conjecture a similar result can be obtained in the multi-stage setting for the TLS 1.3 Diffie–Hellman-based 0-RTT handshake.

[7] We fix a maximum stage M only for ease of notation. Note that M can be arbitrarily large in order to cover protocols where the number of stages is not bounded a-priori. Also note that, for technical convenience, stages and session keys may be "back to back," without further protocol interactions between parties.

- $\mathsf{AUTH} \subseteq \{\mathsf{unauth}, \mathsf{unilateral}, \mathsf{mutual}\}^{\mathsf{M}}$: the set of supported authentication properties (for each stage). We call stages and keys *unauthenticated* if they provide no authentication for either communication partner, *unilaterally authenticated* if they authenticate only the responder (server) side, and *mutually authenticated* if they authenticate both communication partners.

- $\mathsf{USE} \in \{\mathsf{internal}, \mathsf{external}\}^{\mathsf{M}}$: the usage indicator for each stage, where USE_i indicates the usage of the stage-i key. Here, an internal key is used within the key exchange protocol (but possibly also externally), whereas an external key must not be used within the protocol, making the latter potentially amenable to generic composition (see Section 4.6).

- $\mathsf{REPLAY} \in \{\mathsf{replayable}, \mathsf{nonreplayable}\}^{\mathsf{M}}$: the replayability indicator for each stage, where REPLAY_i indicates whether the i-th stage is replayable in the sense that an adversary can easily force identical communication and thus identical session identifiers and keys in this stage (e.g., by re-sending the same data in 0-RTT stages). Note that the adversary, however, should still not able to distinguish such a replayed key from a random one. We remark that, from a security viewpoint, the usage of replayable stages should ideally be small, whereas such stages usually come with an efficiency benefit.

As for the basic Bellare–Rogaway model (see Section 3.1), we denote by $\mathcal{U}$ the set of *identities* (or *users*) used to model the participants in the system, each identified by some $U \in \mathcal{U}$. *Sessions* of a protocol are uniquely identified (on the administrative level of the model) using a *label* $\mathsf{label} \in \mathsf{LABELS} = \mathcal{U} \times \mathcal{U} \times \mathbb{N}$, where $\mathsf{label} = (U, V, k)$ indicates the k-th local session of identity U (the session *owner*) with V as the intended communication *partner*.

In the public-key variant of the model (MSKE), each identity U is associated with a certified *long-term* public key pk_U and secret key sk_U. In the pre-shared secret setting (MS-PSKE), a session instead holds a key index for the pre-shared secret pss (and its unique identifier psid) used. The challenger maintains vectors $\mathsf{pss}_{U,V}$ and $\mathsf{psid}_{U,V}$ of pre-shared secrets created on adversary demand, with the k-th entry indicating the k-th secret, resp. corresponding identifier, shared by parties U and V.

In addition to the long-term keys, parties may in the public-key setting also hold certified *semi-static* key pairs $(\mathsf{sspk}, \mathsf{sssk})$, each identified by a semi-static key identifier sskid.[8] Semi-static keys moreover are associated with some auxiliary data sskaux which may for example carry the data structure in which a party learns the semi-static key (e.g., the `ServerConfiguration` message and other identifiers in TLS 1.3's Diffie–Hellman-based 0-RTT handshake). Finally, a flag $\mathsf{st}_{\mathsf{ssk},\mathsf{sskid}} \in \{\mathsf{fresh}, \mathsf{revealed}\}$ indicates whether a semi-static key has been revealed to the adversary or not. This flag is convenient since in our model semi-static keys are linked to replayable stages, especially to 0-RTT stages (as is common practice), such that we consider the disclosure of such keys inevitably rendering these session keys to be insecure.

For each session, a tuple with the following information is maintained as an entry in the *session list* $\mathsf{List}_{\mathsf{S}}$, where values in square brackets [] indicate the default initial value. Some variables have values for each stage $i \in \{1, \ldots, \mathsf{M}\}$.

- $\mathsf{label} \in \mathsf{LABELS}$: the unique (administrative) session label

- $\mathsf{id} \in \mathcal{U}$: the identity of the session owner

[8] In earlier versions of the multi-stage key exchange model [FG14, DFGS15a], we denoted those keys as "temporary keys." Here, we adopt the more common terminology of "semi-static keys" which is also used in TLS 1.3.

- $\mathsf{pid} \in (\mathcal{U} \cup \{*\})$: the identity of the intended communication partner, where the distinct wildcard symbol '$*$' stands for "unknown identity" and can be set to a specific identity in $\mathcal{U}$ once by the protocol
- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$: the session owner's role in this session
- $\mathsf{auth} \in \mathsf{AUTH}$: the intended authentication type (for each stage) from the set of supported authentication properties AUTH, where auth_i indicates the authentication level in stage i
- $\mathsf{st}_{\mathsf{exec}} \in (\mathsf{RUNNING} \cup \mathsf{ACCEPTED} \cup \mathsf{REJECTED})$: the state of execution [$\mathsf{running}_0$], where $\mathsf{RUNNING} = \{\mathsf{running}_i \mid i \in \mathbb{N} \cup \{0\}\}$, $\mathsf{ACCEPTED} = \{\mathsf{accepted}_i \mid i \in \mathbb{N}\}$, $\mathsf{REJECTED} = \{\mathsf{rejected}_i \mid i \in \mathbb{N}\}$, set to $\mathsf{accepted}_i$ in the moment a session accepts the i-th key, to $\mathsf{rejected}_i$ when the session rejects that key (we assume a session does not continue after a reject in any stage), and to $\mathsf{running}_i$ when a session continues after accepting the i-th key
- $\mathsf{stage} \in \{0, \ldots, \mathsf{M}\}$: the current stage [0], where stage is incremented to i when $\mathsf{st}_{\mathsf{exec}}$ reaches $\mathsf{accepted}_i$ resp. $\mathsf{rejected}_i$
- $\mathsf{sid} \in (\{0,1\}^* \cup \{\bot\})^{\mathsf{M}}$: sid_i [$\bot$] indicates the session identifier in stage i, set once upon acceptance in that stage
- $\mathsf{cid} \in (\{0,1\}^* \cup \{\bot\})^{\mathsf{M}}$: cid_i [$\bot$] indicates the contributive identifier in stage i, may be set several times until acceptance in that stage
- $\mathsf{key} \in (\{0,1\}^* \cup \{\bot\})^{\mathsf{M}}$: key_i [$\bot$] indicates the established session key in stage i, set once upon acceptance in that stage
- $\mathsf{st}_{\mathsf{key}} \in \{\mathsf{fresh}, \mathsf{revealed}\}^{\mathsf{M}}$: $\mathsf{st}_{\mathsf{key},i}$ [fresh] indicates the state of the session key in stage i
- $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}^{\mathsf{M}}$: test indicator tested_i [false], where true means that key_i has been tested

In the public-key (MSKE) variant, $\mathsf{List}_{\mathsf{S}}$ furthermore contains the following entries:

- $\mathsf{sskid}_{\mathsf{id}}$: the key identifier for the semi-static key pair ($\mathsf{sspk}, \mathsf{sssk}$) used by the session owner ($\bot$ if no key is used)
- $\mathsf{sskid}_{\mathsf{pid}}$: the semi-static key identifier for the communication partner ($\bot$ if no key is used)

In the pre-shared secret (MS-PSKE) variant, $\mathsf{List}_{\mathsf{S}}$ instead contains the following extra entries:

- $k \in \mathbb{N}$: the index of the pre-shared secret used in a protocol run with the communication partner
- $\mathsf{pss} \in (\{0,1\}^* \cup \{\bot\})$: the pre-shared secret to be used in the session
- $\mathsf{psid} \in (\{0,1\}^* \cup \{\bot\})$: the pre-shared secret identifier of the pre-shared secret to be used in the session

By convention, adding a not fully specified tuple ($\mathsf{label}, \mathsf{id}, \mathsf{pid}, \mathsf{role}, \mathsf{auth}, \mathsf{sskid}_{\mathsf{id}}, \mathsf{sskid}_{\mathsf{pid}}$) resp. ($\mathsf{label}, \mathsf{id}, \mathsf{pid}, \mathsf{role}, \mathsf{auth}, k, \mathsf{pss}, \mathsf{psid}$) to $\mathsf{List}_{\mathsf{S}}$ sets all other entries to their default value. We furthermore write, e.g., $\mathsf{label.sid}$ as shorthand for the element sid in the tuple with (unique) label label in $\mathsf{List}_{\mathsf{S}}$.

As for the plain Bellare–Rogaway model, we define two distinct sessions label and label' to be *partnered* if both sessions hold the same session identifier, i.e., $\mathsf{label.sid} = \mathsf{label}'\mathsf{.sid} \neq \bot$, and require for correctness that two sessions having a non-tampered joint execution are partnered upon acceptance.

4.4 Adversary Model

We consider a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ which controls the communication between all parties, enabling interception, injection, and dropping of messages. Our adversary model further reflects the advanced security aspects in multi-stage key exchange as outlined in Section 4.2. We conveniently capture admissibility of adversarial interactions and conditions where the adversary trivially loses (such as both revealing and testing the session key in partnered sessions) via a flag lost (initialized to false).

The adversary interacts with the protocol via the following queries.

- $\mathsf{NewSecret}(U, V, k, \mathsf{psid})$: This query is only available in the pre-shared secret (MS-PSKE) variant. Generates a fresh pre-shared secret with identifier psid shared as k-th secret between parties U and V. If there already is a k-th entry in $\mathsf{pss}_{U,V}$ or if psid is already registered for any other pss, return $\perp$. The latter ensures global uniqueness of the psid value. Otherwise, sample pss uniformly at random from the protocol's pre-shared secret space and store pss and psid as the k-th entry in $\mathsf{pss}_{U,V}$ and $\mathsf{pss}_{\mathbf{V},\mathbf{U}}$, resp. in $\mathsf{psid}_{\mathbf{V},\mathbf{U}}$ and $\mathsf{psid}_{\mathbf{V},\mathbf{U}}$.

- $\mathsf{NewSemiStaticKey}(U, \mathsf{sskaux}_{\mathsf{pre}})$: This query is only available in the public-key (MSKE) variant. Generates a new semi-static key pair $(\mathsf{sspk}, \mathsf{sssk})$ for identity U with associated (protocol-defined) auxiliary data sskaux (which might include some adversarially pre-specified parts $\mathsf{sskaux}_{\mathsf{pre}}$) and a (unique) new identifier sskid. Set $\mathsf{st}_{\mathsf{ssk},\mathsf{sskid}} \leftarrow \mathsf{fresh}$ and return the tuple $(\mathsf{sskid}, \mathsf{sspk}, \mathsf{sskaux})$.

- $\mathsf{NewSession}(U, V, \mathsf{role}, \mathsf{auth}, \mathsf{sskid}_U, \mathsf{sskid}_V)$ or $\mathsf{NewSession}(U, V, \mathsf{role}, \mathsf{auth}, k)$: The first query is only used in the public-key (MSKE) variant, the second query only in the pre-shared secret (MS-PSKE) variant. Creates a new session with a (unique) new label label for owner participant identity $\mathsf{id} = U$ with role role, having $\mathsf{pid} = V$ as intended partner (potentially unspecified, indicated by $V = *$) and aiming at authentication type $\mathsf{auth} \in \mathsf{AUTH}$.

  In the public-key variant, sskid_U and sskid_V indicate the semi-static key identifiers used by the owner, resp. intended partner. Either semi-static key identifier can also be left unspecified ($\mathsf{sskid}_U = \perp$, resp. $\mathsf{sskid}_V = \perp$), indicating that the according party does not use such key. In the pre-shared secret variant, k indicates the key index of the shared pss between U and V. If no such pss has been registered, return $\perp$. Otherwise, set $\mathsf{label.pss}$ to pss and $\mathsf{label.psid}$ to the corresponding k-th entry of $\mathsf{psid}_{U,V}$. Add $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{auth}, \mathsf{sskid}_U, \mathsf{sskid}_V)$, resp. $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{auth}, k, \mathsf{pss}, \mathsf{psid})$, to $\mathsf{List}_{\mathsf{S}}$ and return label.

- $\mathsf{Send}(\mathsf{label}, m)$: Sends a message m to the session with label label.

  If there is no tuple with label label in $\mathsf{List}_{\mathsf{S}}$, return $\perp$. Otherwise, run the protocol on behalf of U on message m and return the response and the updated state of execution $\mathsf{label.st}_{\mathsf{exec}}$. As a special case, if $\mathsf{label.role} = \mathsf{initiator}$ and $m = \mathsf{init}$, the protocol is initiated (without any input message).

  If, during the protocol execution, the state of execution changes to $\mathsf{accepted}_i$ for some i, the protocol execution is immediately suspended and $\mathsf{accepted}_i$ is returned as result to the adversary. The adversary can later trigger the resumption of the protocol execution by issuing a special $\mathsf{Send}(\mathsf{label}, \mathsf{continue})$ query. For such a query, the protocol continues as specified, with the party creating the next protocol message and handing it over to the adversary together with the resulting state of execution $\mathsf{st}_{\mathsf{exec}}$. We note that this is necessary to allow the adversary to test such a key, before it may be used immediately in the response and thus cannot be tested anymore to prevent trivial distinguishing attacks.

If the state of execution changes to $\mathsf{label.st_{exec}} = \mathsf{accepted}_i$ for some i and there is a partnered session $\mathsf{label}' \neq \mathsf{label}$ in $\mathsf{List_S}$ (i.e., $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$) with $\mathsf{label}'.\mathsf{st}_{\mathsf{key},i} = \mathsf{revealed}$, then, for key independence, $\mathsf{label.st}_{\mathsf{key},i}$ is set to $\mathsf{revealed}$ as well, whereas for key-dependent security, all $\mathsf{label.st}_{\mathsf{key},i'}$ for $i' \geq i$ are set to $\mathsf{revealed}$. The former corresponds to the case that session keys of partnered sessions should be considered revealed as well, the latter implements that for key dependency all subsequent keys are potentially available to the adversary, too.

If the state of execution changes to $\mathsf{label.st_{exec}} = \mathsf{accepted}_i$ for some i and there is a partnered session $\mathsf{label}' \neq \mathsf{label}$ in $\mathsf{List_S}$ (i.e., $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$) with $\mathsf{label}'.\mathsf{tested}_i = \mathsf{true}$, then set $\mathsf{label.tested}_i \leftarrow \mathsf{true}$ and (only if $\mathsf{USE}_i = \mathsf{internal}$) $\mathsf{label.key}_i \leftarrow \mathsf{label}'.\mathsf{key}_i$. This ensures that, if the partnered session has been tested before, subsequent Test queries for the session are answered accordingly and, in case it is used internally, this session's key key_i is set consistently.[9]

If the state of execution changes to $\mathsf{label.st_{exec}} = \mathsf{accepted}_i$ for some i and the intended communication partner $\mathsf{pid} \neq *$ is corrupted, then set $\mathsf{label.st}_{\mathsf{key},i} \leftarrow \mathsf{revealed}$.

- $\mathsf{Reveal}(\mathsf{label}, i)$: Reveals the session key $\mathsf{label.key}_i$ of stage i in the session with label label.

  If there is no session with label label in $\mathsf{List_S}$ or $\mathsf{label.stage} < i$, then return $\perp$. Otherwise, set $\mathsf{label.st}_{\mathsf{key},i}$ to $\mathsf{revealed}$ and provide the adversary with $\mathsf{label.key}_i$.

  If there is a partnered session label' in $\mathsf{List_S}$ (i.e., $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$) with $\mathsf{label}'.\mathsf{stage} \geq i$, then $\mathsf{label}'.\mathsf{st}_{\mathsf{key},i}$ is set to $\mathsf{revealed}$ as well. This means the i-th session keys of all partnered sessions (if established) are considered revealed too.

  As above, in the case of key-dependent security, since not yet established future keys depend on the revealed key, we cannot ensure their security anymore (neither in this session in question, nor in partnered sessions). Therefore, if $\mathsf{label.stage} = i$, set $\mathsf{label.st}_{\mathsf{key},j} = \mathsf{revealed}$ for all $j > i$, as they depend on the revealed key. For the same reason, if a partnered session label' ($\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$) has $\mathsf{label}'.\mathsf{stage} = i$, then set $\mathsf{label}'.\mathsf{st}_{\mathsf{key},j} = \mathsf{revealed}$ for all $j > i$. Note that if however $\mathsf{label}'.\mathsf{stage} > i$, then keys $\mathsf{label}'.\mathsf{key}_j$ for $j > i$ derived in the partnered session are not considered to be revealed by this query since they have been accepted previously, i.e., prior to key_i being revealed in this query.

- $\mathsf{RevealSemiStaticKey}(\mathsf{sskid})$: This query is only available in the public-key (MSKE) variant. If there exists a semi-static key pair $(\mathsf{sspk}, \mathsf{sssk})$ with identifier sskid, set $\mathsf{st}_{\mathsf{ssk},\mathsf{sskid}} \leftarrow \mathsf{revealed}$, and output sssk. Otherwise, return $\perp$.

  Furthermore, for each session label with $\mathsf{label.sskid_{id}} = \mathsf{sskid}$ or $\mathsf{label.sskid_{pid}} = \mathsf{sskid}$ and all replayable stages $i \in \{1, \ldots, \mathsf{M}\}$ with $\mathsf{REPLAY}_i = \mathsf{replayable}$, set $\mathsf{label.st}_{\mathsf{key},i}$ to $\mathsf{revealed}$. That is, any replayable stage's session key in a session that uses the revealed semi-static key is considered to be disclosed.

- $\mathsf{Corrupt}(U)$ or $\mathsf{Corrupt}(\mathsf{psid})$: The first query is only used in the public-key (MSKE) variant, the second query only in the pre-shared secret (MS-PSKE) variant. Provide the adversary with the corresponding long-term secret, i.e., sk_U (MSKE), resp. pss corresponding to psid (MS-PSKE). No further queries are allowed to sessions owned by U (MSKE), resp. to any session label with $\mathsf{label.psid} = \mathsf{psid}$ (MS-PSKE).

  In the non-forward-secret case, for each session label with $\mathsf{label.id} = U$ or $\mathsf{label.pid} = U$ (MSKE), resp. holding $\mathsf{label.psid} = \mathsf{psid}$ (MS-PSKE), and for all $i \in \{1, \ldots, \mathsf{M}\}$, set

[9] Note that for internal keys this implicitly assumes the following property of the later-defined Match security: Whenever two partnered sessions both accept a key in some stage, these keys will be equal.

$\mathsf{label.st}_{\mathsf{key},i}$ to $\mathsf{revealed}$. In this case, all (previous and future) session keys are considered to be disclosed.

In the case of stage-j forward secrecy, $\mathsf{st}_{\mathsf{key},i}$ of such sessions label is instead set to revealed only if $i < j$ or if $i > \mathsf{stage}$. This means that session keys before the j-th stage (where forward secrecy kicks in) as well as keys that have not yet been established are potentially disclosed.

Independent of the forward secrecy aspect, in the case of key-dependent security, setting the relevant key states to $\mathsf{revealed}$ for some stage i is done by internally invoking $\mathsf{Reveal}(\mathsf{label}, i)$, ignoring the response and also the restriction that a call with $i > \mathsf{stage}$ would immediately return $\perp$. This ensures that follow-up revocations of keys that depend on the revoked keys are carried out correctly.

- $\mathsf{Test}(\mathsf{label}, i)$: Tests the session key of stage i in the session with label label. In the security game this oracle is given a uniformly random test bit b_{test} as state which is fixed throughout the game.

  If there is no session with label label in $\mathsf{List}_{\mathsf{S}}$ or if $\mathsf{label.st}_{\mathsf{exec}} \neq \mathsf{accepted}_i$ or $\mathsf{label.tested}_i = \mathsf{true}$, return $\perp$. If there is a partnered session label' in $\mathsf{List}_{\mathsf{S}}$ (i.e., $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$) with $\mathsf{label}'.\mathsf{st}_{\mathsf{exec}} \neq \mathsf{accepted}_i$, set the 'lost' flag to $\mathsf{lost} \leftarrow \mathsf{true}$. This ensures that keys can only be tested once and if they have just been accepted but not used yet, including ensuring any partnered session that may have already established this key has not used it.

  If $\mathsf{label.auth}_i = \mathsf{unauth}$ or if $\mathsf{label.auth}_i = \mathsf{unilateral}$ and $\mathsf{label.role} = \mathsf{responder}$, but there is no session label' (for $\mathsf{label} \neq \mathsf{label}'$) in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{label.cid}_i = \mathsf{label}'.\mathsf{cid}_i$, then set $\mathsf{lost} \leftarrow \mathsf{true}$. This ensures that having an honest contributive partner is a prerequisite for testing unauthenticated stages, resp. the responder sessions in a unilaterally authenticated stage.[10]

  Otherwise, set $\mathsf{label.tested}_i$ to true. If the test bit b_{test} is 0, sample a key $K \xleftarrow{\$} \mathcal{D}$ at random from the session key distribution $\mathcal{D}$. If $b_{\mathsf{test}} = 1$, let $K \leftarrow \mathsf{label.key}_i$ be the real session key. If $\mathsf{USE}_i = \mathsf{internal}$ (i.e., the tested i-th key is indicated as being used internally), set $\mathsf{label.key}_i \leftarrow K$, i.e., we substitute an *internally* used session key by the random and independent test key K which is also used for consistent future deployments *within* the key exchange protocol. In contrast, *externally used* session keys are not replaced by random ones, the adversary only receives the real (in case $b_{\mathsf{test}} = 1$) or random (in case $b_{\mathsf{test}} = 0$) key. This distinction between internal and external keys for Test queries emphasizes that external keys are not supposed to be used within the key exchange (and hence there is no need to register the tested random key in the protocol's session key field) while internal keys will be used (and hence the tested random key must be deployed in the remaining protocol steps for consistency).

  Moreover, if there exists a partnered session label' which has also just accepted the i-th key (i.e., $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$ and $\mathsf{label.st}_{\mathsf{exec}} = \mathsf{label}'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}_i$), then also set $\mathsf{label}'.\mathsf{tested}_i \leftarrow \mathsf{true}$ and (only if $\mathsf{USE}_i = \mathsf{internal}$) $\mathsf{label}'.\mathsf{key}_i \leftarrow \mathsf{label.key}_i$ to ensure consistency (of later tests and (internal) key usage) in the special case that both label and label' are in state $\mathsf{accepted}_i$ and, hence, either of them can be tested first.

  Return K.

[10] Note that $\mathsf{List}_{\mathsf{S}}$ entries are only created for honest sessions, i.e., sessions generated by $\mathsf{NewSession}$ queries.

4.5 Security of Multi-Stage Key Exchange Protocols

As in the formalization of the Bellare–Rogaway key exchange model by Brzuska et al. [BFWW11, Brz13] (cf. Section 3.1), we model security according to two games, one for key indistinguishability, and one for session matching. The former is the classical notion of random-looking keys, refined under the term Multi-Stage security according to the advanced security aspects for multi-stage key exchange: key (in)dependence, (stage-j) forward secrecy, different authentication modes, and replayability. The Match property complements this notion by guaranteeing that the specified session identifiers sid effectively match the partnered sessions, and is likewise adapted to the multi-stage setting.

4.5.1 Match Security

The notion of Match security ensures soundness of the session identifiers sid, i.e., that they properly identify partnered sessions in the sense that

1. sessions with the same session identifier for some stage hold the same key at that stage,
2. sessions with the same session identifier for some stage agree on that stage's authentication level,
3. sessions with the same session identifier for some stage share the same contributive identifier at that stage,
4. sessions are partnered with the intended (authenticated) participant, and for mutual authentication based on pre-shared secrets share the same key index,
5. session identifiers do not match across different stages, and
6. at most two sessions have the same session identifier at any non-replayable stage.

The Match security game $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$ thus is defined as follows.

Definition 4.1 (Match security)**.** *Let* KE *be a multi-stage key exchange protocol with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *and* $\mathcal{A}$ *a PPT adversary interacting with* KE *via the queries defined in Section 4.4 in the following game* $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$*:*

Setup. *In the public-key variant (MSKE), the challenger generates long-term public/private-key pairs for each participant* $U \in \mathcal{U}$*.*

Query. *The adversary* $\mathcal{A}$ *receives the generated public keys (MSKE) and has access to the queries* NewSecret*,* NewSemiStaticKey*,* NewSession*,* Send*,* Reveal*,* RevealSemiStaticKey*,* Corrupt*, and* Test*.*

Stop. *At some point, the adversary stops with no output.*

We say that $\mathcal{A}$ *wins the game, denoted by* $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} = 1$*, if at least one of the following conditions holds:*

1. *There exist two distinct labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \perp$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$*,* $\mathsf{label.st}_{\mathsf{exec}} \neq \mathsf{rejected}_i$*, and* $\mathsf{label'.st}_{\mathsf{exec}} \neq \mathsf{rejected}_i$*, but* $\mathsf{label.key}_i \neq \mathsf{label'.key}_i$*. (Different session keys in some stage of partnered sessions.)*
2. *There exist two distinct labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \perp$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$*, but* $\mathsf{label.auth}_i \neq \mathsf{label'.auth}_i$*. (Different authentication types in some stage of partnered sessions.)*

3. *There exist two distinct labels* label, label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \perp$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$, *but* $\mathsf{label.cid}_i \neq \mathsf{label'.cid}_i$ *or* $\mathsf{label.cid}_i = \mathsf{label'.cid}_i = \perp$. *(Different or unset contributive identifiers in some stage of partnered sessions.)*

4. *There exist two distinct labels* label, label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \perp$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$, $\mathsf{label.auth}_i = \mathsf{label'.auth}_i \in \{\mathsf{unilateral}, \mathsf{mutual}\}$, $\mathsf{label.role} = \mathsf{initiator}$, *and* $\mathsf{label'.role} = \mathsf{responder}$, *but* $\mathsf{label.pid} \neq \mathsf{label'.id}$ *or (only if* $\mathsf{label.auth}_i = \mathsf{mutual}$*)* $\mathsf{label.id} \neq \mathsf{label'.pid}$ *or (only for MS-PSKE and if* $\mathsf{label.auth}_i = \mathsf{mutual}$*)* $\mathsf{label}.k \neq \mathsf{label'}.k$. *(Different intended authenticated partner or (only MS-PSKE) different key indices in mutual authentication.)*

5. *There exist two (not necessarily distinct) labels* label, label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_j \neq \perp$ *for some stages* $i, j \in \{1, \ldots, \mathsf{M}\}$ *with* $i \neq j$. *(Different stages share the same session identifier.)*

6. *There exist three pairwise distinct labels* label, label′, label″ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i = \mathsf{label''.sid}_i \neq \perp$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$ *with* $\mathsf{REPLAY}_i = \mathsf{nonreplayable}$. *(More than two sessions share the same session identifier in a non-replayable stage.)*

We say KE *is* Match-secure *if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} = 1\right].$$

4.5.2 Multi-Stage Security

The second and core notion, Multi-Stage security, captures Bellare–Rogaway-like key secrecy in the multi-stage setting as follows.

Definition 4.2 (Multi-Stage security)**.** *Let* KE *be a multi-stage key exchange protocol with key distribution* $\mathcal{D}$ *and properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$, *and* $\mathcal{A}$ *a PPT adversary interacting with* KE *via the queries defined in Section 4.4 in the following game* $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}$*:*

Setup. *The challenger chooses the test bit* $b_{\mathsf{test}} \xleftarrow{\$} \{0,1\}$ *at random and sets* $\mathsf{lost} \leftarrow \mathsf{false}$. *In the public-key variant (MSKE), it furthermore generates long-term public/private-key pairs for each participant* $U \in \mathcal{U}$.

Query. *The adversary* $\mathcal{A}$ *receives the generated public keys (MSKE) and has access to the queries* NewSecret, NewSemiStaticKey, NewSession, Send, Reveal, RevealSemiStaticKey, Corrupt, *and* Test. *Note that such queries may set* lost *to* true.

Guess. *At some point,* $\mathcal{A}$ *stops and outputs a guess* b.

Finalize. *The challenger sets the 'lost' flag to* $\mathsf{lost} \leftarrow \mathsf{true}$ *if there exist two (not necessarily distinct) labels* label, label′ *and some stage* $i \in \{1, \ldots, \mathsf{M}\}$ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i$, $\mathsf{label.st}_{\mathsf{key},i} = \mathsf{revealed}$, *and* $\mathsf{label'.tested}_i = \mathsf{true}$. *(Adversary has tested and revealed the key of some stage in a single session or in two partnered sessions.)*

We say that $\mathcal{A}$ *wins the game, denoted by* $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} = 1$, *if* $b = b_{\mathsf{test}}$ *and* $\mathsf{lost} = \mathsf{false}$. *Note that the winning condition is independent of key dependency, forward secrecy, and authentication properties of* KE, *as those are directly integrated in the affected (*Reveal *and* Corrupt*) queries and the finalization step of the game; for example,* Corrupt *is defined differently for non-forward-secrecy versus stage-*j *forward secrecy.*

We say KE *is* Multi-Stage-secure *in a key-dependent resp. key-independent and non-forward-secret resp. stage-j-forward-secret manner with concurrent authentication types* AUTH*, key usage* USE*, and replayability property* REPLAY *if* KE *is* Match*-secure and for all PPT adversaries $\mathcal{A}$ the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} = 1\right] - \frac{1}{2}.$$

4.6 Composition

Key exchange protocols would be of limited use if applied in isolation; in general the derived keys are meant to be deployed in a follow-up (or overall) protocol. The most common application is of course the encryption (and authentication) of data sent between the two involved parties within a (cryptographic) channel protocol, with the QUIC transport and TLS record protocols being prime examples. The TLS 1.3 handshakes additionally derive further keys for different purposes, namely for later resumption of a session and for exporting extra key material from a connection. For both, the key usage in the cryptographic channel as well as the usage for other purposes, it is desirable to modularize the security analysis, treating key exchange and the composed protocol(s) independently and then obtaining overall security guarantees for the combined execution.

Ideally, one would hence like to see a composition result for Multi-Stage-secure key exchange protocols in the sense that such protocols—potentially under some condition—can be securely composed with arbitrary symmetric-key protocols, as has been shown for the case of Bellare–Rogaway-secure key exchange protocols by Brzuska et al. [BFWW11]. In this section, we prove that indeed secure composition with arbitrary symmetric-key protocols is possible for a specific flavor of Multi-Stage-secure protocols, namely those that provide key independence and stage-j forward secrecy (as well as a technical notion of multi-stage session matching we define below), when composed with a symmetric-key protocol at a forward-secret, external, and non-replayable stage.

4.6.1 Preliminaries

In order to reason about composition of key exchange and symmetric-key protocol games, we employ the syntax for composed games as well as the notion of session matching introduced by Brzuska et al. [BFWW11, Brz13]. We adapt their syntax and notions to the multi-stage setting. We furthermore extend them to also capture, beyond mutual authentication, composition for unilaterally authenticated and unauthenticated keys, as well as session matching with non-public partnering.

Composed games for multi-stage key exchange. Let G_{KE} be a game modeling security for a (multi-stage) key exchange protocol KE, and G_{Π} a security game for some symmetric-key protocol Π. Fix some stage i for the moment and keys derived in this stage only; the composition with protocols run on keys for other stages will follow from this via the possibility to reveal such keys via the Reveal query.

We define $G_{\mathsf{KE}_i;\Pi}$ as the security game for the composition $\mathsf{KE}_i;\Pi$ of KE and Π as follows: Whenever a session key key_i is accepted *in stage i* of a session of KE where each of the two sessions involved either are authenticated or contributed honestly to the derived key[11], this

[11] More formally, we consider stage-i keys which are accepted in a session label that either has an authenticated communication partner (i.e., $\mathsf{label.auth}_i = \mathsf{mutual}$ or $\mathsf{label.auth}_i = \mathsf{unilateral}$ and $\mathsf{label.role} = \mathsf{initiator}$) or an honest contributing partnered session (i.e., there exists a session label′ with $\mathsf{label.cid}_i = \mathsf{label'.cid}_i$).

key key_i is registered as a new key in the symmetric-key protocol game G_Π, allowing the adversary to run Π-sessions with this key (and all previously registered keys). Observe that compositional security can obviously only be guaranteed when the adversary does not know the derived session key, which we require the key exchange protocol to ensure whenever both sides of the key exchange are either authenticated or honest in their contribution. In particular, if a session key is derived in a key exchange involving an unauthenticated party whose key contribution was not simulated by the challenger, we must expect that the adversary controls this party to an extent where it holds the derived session key—and hence cannot require any security property of the symmetric-key protocol to hold for such a session key.

In $G_{\mathsf{KE}_i;\Pi}$, the adversary's task is to break the security of Π by winning in the subgame G_Π, given access to both the queries of G_{KE} and G_Π, which the composed game essentially just relays to the appropriate subgame. Exceptions to this are the key registration queries of G_Π (that are only executed by the composed game to register stage-i keys within G_Π whenever such a key has been accepted), the Reveal query of G_{KE} (which the adversary is not allowed to query *for stage-i keys* in the composed game[12], as session key compromise for these keys is—if at all—captured in G_Π), and the Test query of G_{KE} (being only of administrative purpose for G_{KE}). The adversary wins in the composed game, if it, via its queries, succeeds in the subgame G_Π.

Multi-stage session matching. As established by Brzuska et al. [BFWW11], session matching is both a necessary and sufficient condition for the composition of Bellare–Rogaway-secure key exchange and generic symmetric-key protocols. In their definition, a key exchange protocol KE allows for session matching if there exists an efficient algorithm that, when eavesdropping on the communication between an arbitrary adversary $\mathcal{A}$ and the security game G_{KE}, is able to deduce which sessions are partnered at each point of the communication. They moreover observe that such a matching might not be (efficiently) computable in certain cases, e.g., if the key exchange messages are encrypted using a (publicly) re-randomizable cipher, but partnering is defined over the unencrypted messages.

The latter restriction becomes particularly relevant in the multi-stage setting, as key exchange protocols may—and TLS 1.3 does—use keys of previous stages to encrypt later stages' messages. In such cases, session matching based on the public transcript may not be feasible anymore; this especially holds for the case of TLS 1.3. We can however leverage that key independence is already a prerequisite for composition in the multi-stage setting and hence, when targeting the keys of a certain stage, revealing the keys of previous stages is of no harm in the key exchange game. Therefore, we can strengthen session matching in the multi-stage setting to obtain also the session keys key_j for all stages $j < i$ when determining the partnering for stage i. We moreover extend session matching to comprise not only the session identifiers but also the newly introduced contributive identifiers.

Formally, we define *multi-stage session matching* as follows.

Definition 4.3 (Multi-stage session matching algorithm)**.** *A* multi-stage session matching algorithm $\mathcal{M}$ *for a key exchange protocol* KE *is an efficient algorithm for which the following holds for any adversary* $\mathcal{A}$ *interacting in the* $\mathsf{Multi\text{-}Stage}$ *security game* $G_{\mathsf{KE},\mathcal{A}}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}$ *of* KE*. On input a stage i, the public parameters of the game, an ordered list of all queries made by* $\mathcal{A}$ *and responses from* $G_{\mathsf{KE},\mathcal{A}}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}$ *at any point of the game execution, and, for all sessions, a list of all stage-j keys (for any $j < i$) accepted at this point,* $\mathcal{M}$ *outputs two lists of pairs of all sessions in stage i, the first list containing exactly those pairs sharing the same session identifier* sid_i

[12] Note however that keys in stages different from i, not being used for Π, are still accessible via Reveal queries in $G_{\mathsf{KE}_i;\Pi}$, which makes our result also cover *concurrent composition* with one (or several) of such protocols using the keys from multiple stages.

(i.e., being partnered), and the second list exactly those pairs sharing the same contributive identifier cid_i *at this point of the game execution.*

If such an algorithm exists for a key exchange protocol KE*, we say that* KE *allows for an* efficient multi-stage session matching.

4.6.2 Compositional Security

We are now able to state our composition result for multi-stage key exchange protocols: The composition $\mathsf{KE}_i;\Pi$ of a multi-stage key exchange protocol KE with an arbitrary symmetric-key protocol Π employing the stage-i session keys of KE is secure if the key exchange is Multi-Stage-secure providing *key independence*, *stage-j forward secrecy* (for $j \leq i$), *multi-stage session matching*, and the stage-i keys are *external* and *non-replayable*.

Theorem 4.4 (Multi-stage composition)**.** *Let* KE *be a* Multi-Stage*-secure key exchange protocol (in the public-key or preshared-secret setting) providing key independence and stage-j forward secrecy with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *and key distribution* $\mathcal{D}$*, and that allows for efficient multi-stage session matching. Let* Π *be a symmetric-key protocol that is secure w.r.t. some game* G_Π *and has a key generation algorithm that outputs keys with distribution* $\mathcal{D}$*. Then the composition* $\mathsf{KE}_i;\Pi$ *for any external and non-replayable stage* $i \geq j$ *(i.e.,* $\mathsf{REPLAY}_i = \mathsf{nonreplayable}$ *and* $\mathsf{USE}_i = \mathsf{external}$*) is secure w.r.t. the composed security game* $G_{\mathsf{KE}_i;\Pi}$*. Formally, for any efficient adversary* $\mathcal{A}$ *against* $G_{\mathsf{KE}_i;\Pi}$ *there exist efficient algorithms* $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ *such that*

$$\mathsf{Adv}^{G_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE},\mathcal{B}_1} + n_s \cdot \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}_2} + \mathsf{Adv}^{G_\Pi}_{\Pi,\mathcal{B}_3},$$

where n_s *is the maximum number of sessions in the key exchange game.*

Proof of Theorem 4.4. The proof proceeds along the lines of the Bellare–Rogaway composition result by Brzuska et al. [BFWW11].

As a technical prerequisite, we ensure that the key exchange protocol KE in the composed game $G_{\mathsf{KE}_i;\Pi}$ always outputs the same key key_i for two partnered sessions in stage i. This basic property is given by Match security (which is subsumed under requiring Multi-Stage security from KE) and hence we can easily turn an adversary $\mathcal{A}$ that triggers different keys to be output in partnered sessions in the key exchange part of $G_{\mathsf{KE}_i;\Pi}$ into an adversary $\mathcal{B}_1$ against Match security. Observe that $\mathcal{B}_1$ can simply relay all oracle queries $\mathcal{A}$ makes for the KE subgame to its own oracles (note that $\mathcal{A}$ is not given access to a Test query in $G_{\mathsf{KE}_i;\Pi}$). Furthermore, $\mathcal{B}_1$ simulates the Π subgame on its own according to the $G_{\mathsf{KE}_i;\Pi}$ definition. Providing a correct simulation for $\mathcal{A}$, algorithm $\mathcal{B}_1$ always wins if $\mathcal{A}$ makes two partnered sessions output different keys in stage i; hence, we can from this point on assume that partnered sessions agree on their derived keys.

On a high level, we now first replace the derived session keys, one at a time, by a randomly chosen key from $\mathcal{D}$ and show that an adversary able to distinguish each of these replacements can be turned into an efficient Multi-Stage adversary against KE. After all keys have been replaced by random ones, the subgame G_Π is then independent of the key exchange protocol as the now randomly chosen final stage-i keys are not used within the key exchange. Hence, breaking the composed game at this point immediately translates to breaking the symmetric-key protocol game.

The first part of the proof is a hybrid argument. Let $G^n_{\mathsf{KE}_i;\Pi}$ denote a game that behaves like $G_{\mathsf{KE}_i;\Pi}$ (with partnered sessions agreeing on the derived key), except that for the first n accepting sessions in stage i where the key is registered in the symmetric-key protocol subgame (i.e., where the communication partner is either authenticated or an honest partnered session exists), instead of the real session key key_i a randomly chosen $\mathsf{key}'_i \xleftarrow{\$} \mathcal{D}$ is registered in G_Π. Obviously,

$G^0_{\mathsf{KE}_i;\Pi} = G_{\mathsf{KE}_i;\Pi}$ while $G^{n_s}_{\mathsf{KE}_i;\Pi}$ (for n_s being the maximum number of sessions) denotes the game where all keys used in the Π subgame are chosen at random from $\mathcal{D}$. As we establish in Lemma 4.5 below, we have that both games are indistinguishable due to the Multi-Stage security of KE and, for $\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}_2}$ being the maximal advantage of any of the hybrid reductions $\mathcal{B}$ from Lemma 4.5, it holds that

$$\mathsf{Adv}^{G^0_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} \leq \mathsf{Adv}^{G^{n_s}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} + n_s \cdot \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}_2}.$$

In $G^{n_s}_{\mathsf{KE}_i;\Pi}$ only randomly chosen keys, independent of KE, are used in the symmetric-key protocol subgame G_Π. This allows us to bound, in Lemma 4.6 below, the advantage of $\mathcal{A}$ in $G^{n_s}_{\mathsf{KE}_i;\Pi}$ by the advantage of an adversary $\mathcal{B}_3$ directly breaking the protocol security game G_Π.

Finally, the initial assumption that Π is secure w.r.t. G_Π then allows us to conclude that $\mathsf{KE};\Pi$ is secure w.r.t. $G_{\mathsf{KE}_i;\Pi}$. □

We first establish the hybrid argument in the proof of Theorem 4.4.

Lemma 4.5. *Let* KE *be a* Multi-Stage*-secure key exchange protocol (in the public-key or preshared-secret setting) providing key independence and stage-j forward secrecy with properties* (M, AUTH, USE, REPLAY) *and key distribution $\mathcal{D}$, that allows for efficient multi-stage session matching, and where partnered sessions in stage i always agree on the derived session key. Let Π be a symmetric-key protocol that is secure w.r.t. some game G_Π and has a key generation algorithm that outputs keys with distribution $\mathcal{D}$. Then for any external and non-replayable stage $i \geq j$, all $n = 1, \ldots, n_s$, and any efficient adversary $\mathcal{A}$ there exists an efficient algorithm $\mathcal{B}$ such that*

$$\left|\mathsf{Adv}^{G^{n-1}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} - \mathsf{Adv}^{G^{n}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}}\right| \leq \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}},$$

where n_s is the maximum number of sessions in the key exchange game.

We provide $\mathcal{B}$ with n as auxiliary input for simplicity but note that letting $\mathcal{B}$ pick n at random in $[1, n_s]$ suffices to prove the hybrid argument.

Let us remark that each hybrid step makes use of only a single Test query in its reduction to the Multi-Stage security of KE. We expect that the overall bound could be improved by the factor of n_s introduced by the hybrid argument when leveraging the multiple Test queries available in the Multi-Stage game to transition from $G^0_{\mathsf{KE}_i;\Pi}$ to $G^{n_s}_{\mathsf{KE}_i;\Pi}$ directly.

Proof of Lemma 4.5. The task is to construct an algorithm $\mathcal{B}$ given n and using the adversary $\mathcal{A}$ against $G_{\mathsf{KE}_i;\Pi}$ such that, if $\mathcal{A}$ is able to distinguish (by a a non-negligible advantage difference) between $G^{n-1}_{\mathsf{KE}_i;\Pi}$ and $G^{n}_{\mathsf{KE}_i;\Pi}$, then $\mathcal{B}$ has non-negligible advantage in $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$.

In order to simulate $G_{\mathsf{KE}_i;\Pi}$ for $\mathcal{A}$, algorithm $\mathcal{B}$ basically forwards all KE-related queries to its Multi-Stage game as described below while answering queries to the G_Π subgame on its own (using the established stage-i keys from the key exchange). For administrative purposes, $\mathcal{B}$ keeps two mappings. The first one, $\mathsf{SDATA}\colon \mathsf{LABELS} \to \{\mathsf{initiator}, \mathsf{responder}\} \times \{\mathsf{unauth}, \mathsf{unilateral}, \mathsf{mutual}\} \times [\mathcal{D}]^{i-1}$, stores the role, the i-th stage's authentication level, and the session keys for all stages $j < i$ of each key exchange session. The second one, $\mathsf{SKEY}\colon \mathsf{LABELS} \to [\mathcal{D}]$, stores the key value for each session whose stage-i key was registered in G_Π. Morever, $\mathcal{B}$ keeps a counter c, initialized as $c = 0$, indicating the number of session keys replaced by random values so far. Algorithm $\mathcal{B}$ handles queries by $\mathcal{A}$ to the key exchange subgame as follows (recall that there $\mathcal{A}$ does not have access to the Test oracle of G_{KE} in $G_{\mathsf{KE}_i;\Pi}$):

- NewSecret, NewSemiStaticKey, NewSession, Reveal, RevealSemiStaticKey, and Corrupt queries are forwarded to $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ and the responses sent back to $\mathcal{A}$. For any NewSession call,

$\mathcal{B}$ puts the issued label together with the session's specified role and authentication level for stage i into the map SDATA.

Observe that the approach to simply forward Reveal, RevealSemiStaticKey, and Corrupt queries is sound and in particular does not infringe with a later test query on a stage-i key (see below). For the former, KE being key-independent implies that $\mathsf{Reveal}(\mathsf{label}, i')$ queries for stages $i' \neq i$, which are allowed in the composed game, never affect the security of session keys in stage i. For the latter, the stage-j forward secrecy of KE (for $j \leq i$) and non-replayability of stage i ensure that session keys in stage i are not affected by Corrupt or RevealSemiStaticKey queries.

- $\mathsf{Send}(\mathsf{label}, m)$ queries are forwarded to $G_{\mathsf{KE},\mathcal{B}}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}$ as well and the responses sent back to $\mathcal{A}$. Additionally, if session label in $G_{\mathsf{KE},\mathcal{B}}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}$ changes to an accepting state $\mathsf{accepted}_j$ for $j < i$ due to such a query, $\mathcal{B}$ issues a query $\mathsf{Reveal}(\mathsf{label}, j)$ and stores the resulting session key key_j in the map SDATA. Note that, again by key independence, this Reveal query does not affect the session's stage-i key.

  The most important case is when session label changes to state $\mathsf{accepted}_i$. Here, $\mathcal{B}$ first of all invokes the efficient multi-stage session matching algorithm on the queries and responses $\mathcal{A}$ posed to the subgame G_{KE} and all established session keys for stages $j < i$ (which $\mathcal{B}$ has stored in SDATA), in order to obtain all sessions which are partnered and those which agree on the contributive identifier in stage i.

  In case label is partnered with some other session label' and $\mathsf{SKEY}(\mathsf{label}')$ is set, $\mathcal{B}$ sets this key value also as $\mathsf{SKEY}(\mathsf{label})$ and provides $\mathcal{A}$ with a handle for $\mathsf{SKEY}(\mathsf{label})$ in G_Π. Recall that by assumption two accepting partnered sessions always establish identical session keys (a property we ensure through Match security in the proof of Theorem 4.4).

  Otherwise, $\mathcal{B}$ checks whether the conditions for registering the resulting key in the subgame G_Π are satisfied, namely whether session label either has an authenticated communication partner (i.e., if $\mathsf{label.auth}_i = \mathsf{mutual}$ or $\mathsf{label.auth}_i = \mathsf{unilateral}$ and $\mathsf{label.role} = \mathsf{initiator}$, which $\mathcal{B}$ looks up in its map SDATA) or has an honest contributing partnered session (i.e., the session matching outputs a session label' with $\mathsf{label.cid}_i = \mathsf{label}'.\mathsf{cid}_i$). If this is the case, $\mathcal{B}$ increments the counter c and provides $\mathcal{A}$ with an identifier for $\mathsf{SKEY}(\mathsf{label})$ in G_Π, where $\mathsf{SKEY}(\mathsf{label})$ is computed depending on the counter c:

  - If $c < n$, then sample $\mathsf{SKEY}(\mathsf{label}) \xleftarrow{\$} \mathcal{D}$ at random.
  - If $c = n$, then issue a $\mathsf{Test}(\mathsf{label}, i)$ query and store the resulting value in $\mathsf{SKEY}(\mathsf{label})$.
  - If $c > n$, then issue a $\mathsf{Reveal}(\mathsf{label}, i)$ query and store the resulting value in $\mathsf{SKEY}(\mathsf{label})$.

Note that $\mathcal{B}$ first checking for partnered sessions in stage i ensures that it, if at all, only tests the first session accepting a key (avoiding the according 'lost'-flag penalty in the Test query) and never both tests and reveals a key in two partnered sessions (satisfying the finalize condition of the Multi-Stage definition). Moreover, as the compositional game as well as $\mathcal{B}$ only register session keys for which each of the communication partners in the key exchange is either authenticated or contributed honestly, we never test a session with an unauthenticated peer and no honest contributive partnered (satisfying the according conditions in the Test query). Therefore, $\mathcal{B}$ will never cause the 'lost' flag to be set in its $G_{\mathsf{KE},\mathcal{B}}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}$ game. Furthermore, as stage i keys in KE are external, for stage i the tested random key in the Test query of our model (for $b_{\mathsf{test}} = 0$) does not replace the actual key in subsequent protocol steps. Determining the c-th stage-i key established via a Test query hence does not infringe with a correct simulation of $G_{\mathsf{KE}_i;\Pi}$.

When $\mathcal{A}$ terminates, $\mathcal{B}$ stops as well and outputs 1 if $\mathcal{A}$ has won in the composed game (i.e., in the G_Π subgame that $\mathcal{B}$ simulates on its own) and 0 otherwise. That way, if the Test query made by $\mathcal{B}$ returns the real session key, $\mathcal{B}$ perfectly simulates $G^{n-1}_{\mathsf{KE}_i;\Pi}$ for $\mathcal{A}$, whereas, if a random key is returned, $\mathcal{B}$ perfectly simulates $G^{n}_{\mathsf{KE}_i;\Pi}$. Since $\mathcal{B}$ never causes $\mathsf{lost} = 1$ in its game we can hence bound the advantage difference between $\mathsf{Adv}^{G^{n-1}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}}$ and $\mathsf{Adv}^{G^{n}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}}$ by the advantage of $\mathcal{B}$ in winning the game $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$:

$$\left|\mathsf{Adv}^{G^{n-1}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} - \mathsf{Adv}^{G^{n}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}}\right| \leq \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}. \qquad \square$$

It remains to show how an adversary in the hybrid game $G^{n_s}_{\mathsf{KE}_i;\Pi}$, where all composed stage-i session keys in the G_Π subgame are chosen at random and independent of the key exchange (as the according stage i is external), can be reduced to an adversary in security game G_Π of the symmetric-key protocol.

Lemma 4.6. *Let* KE *be a multi-stage key exchange protocol with stage* i *being external and where partnered sessions in stage* i *always agree on the derived session key. Let* Π *be a symmetric-key protocol that is secure w.r.t. some game* G_Π *and has a key generation algorithm that outputs keys with distribution* $\mathcal{D}$*. Let* n_s *be the maximum number of sessions in* $G_{\mathsf{KE}_i;\Pi}$*. Then for any efficient adversary* $\mathcal{A}$ *there exists an efficient algorithm* $\mathcal{C}$ *such that*

$$\mathsf{Adv}^{G^{n_s}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} \leq \mathsf{Adv}^{G_\Pi}_{\Pi,\mathcal{C}}.$$

Proof of Lemma 4.6. We let algorithm $\mathcal{C}$ simulate the entire composed game $G^{n_s}_{\mathsf{KE}_i;\Pi}$ for $\mathcal{A}$, computing the outputs of the key exchange subgame on its own while forwarding any Π-related query to its game G_Π. This is possible, as the keys established in the key exchange stage i are external (i.e., unused in KE), hence independent of the protocol part, and thus $\mathcal{C}$ is indeed able to provide a perfect simulation for $\mathcal{A}$. In the end, if $\mathcal{A}$ wins in the simulated game, $\mathcal{C}$ will have won in its game G_Π as well, establishing the desired equation.

Formally, $\mathcal{C}$ only has to handle Send queries to the key exchange game in a special way. Although all session keys used in the protocol stage are uniformly distributed, $\mathcal{C}$ needs to distinguish two cases when a session key is accepted in the key exchange:

- If the accepting session is partnered, $\mathcal{C}$ instructs G_Π to register the same key as for the partnered session and returns the according key identifier to $\mathcal{A}$.
- Otherwise, $\mathcal{C}$ simply queries G_Π for an identifier of a new (randomly distributed) key chosen by G_Π, which it relays to $\mathcal{A}$.

All other queries are handled by $\mathcal{C}$ in an unmodified way, either by simulating them on its own (in the case of key exchange queries) or by forwarding them to G_Π (in the case of protocol queries).

As G_Π samples keys randomly and $\mathcal{C}$ ensures consistency in the cases of partnered sessions, its simulation of $G^{n_s}_{\mathsf{KE}_i;\Pi}$ for $\mathcal{A}$ is perfect. Since $\mathcal{C}$ forwards all protocol queries of $\mathcal{A}$ unaltered to G_Π, if $\mathcal{A}$ succeeds in the composed game, $\mathcal{C}$ wins in G_Π. $\square$

On the conditions for multi-stage composition. Our composition theorem for multi-stage key exchange protocols is conditioned on a number of properties of the key exchange protocol being composed (recall that there however are no conditions on the composed symmetric-key protocol). After seeing the proof (technique) for the theorem, let us comment on these conditions and augment them with some higher-level rationale.

Forward secrecy. In our hybrid argument in the proof of Theorem 4.4, we depend on the forward secrecy of stage i at which instances of the protocol Π are spawned. More precisely, this property ensures that $\mathsf{Corrupt}$ queries of adversary $\mathcal{A}$ do not affect our simulation of instances of the symmetric-key protocol Π that have already been spawned, thus allowing us to gradually key these protocols with a random instead of the real key. This corresponds to the intuition that, if some of the keys composed into the symmetric-key protocol are later compromised, security of that protocol can in general not be achieved anymore.

We remark that Brzuska [Brz13] describes an avenue towards (restricted) composition for non–forward-secret Bellare–Rogaway-style key exchange protocols. More concretely, if the security notion for the composed symmetric-key protocol is a single-session game (or reducible to such), composition for non–forward-secret key exchange becomes possible as no other keys than the single-session challenge key is registered with the symmetric-key protocol. We conjecture that a similar result can be obtained for multi-stage composition, but restrict our focus here on composition with general symmetric-key protocols.

Key independence. In the proof of our composition theorem, key independence of the multi-stage key exchange guarantees that Reveal queries for session keys $\mathsf{key}_{i'}$ of stages $i' \neq i$ do not affect the session keys in stage i, that we are gradually replacing by random ones. Key independence furthermore enables non-public session matching by providing previous stages' session keys to the matching algorithm to, e.g., decide partnering even on encrypted transcripts (a beneficial feature to argue composition for TLS 1.3).

Indeed, if the key exchange would be key-dependent, revealing a session key of stage $i' < i$ before the key of stage $i' + 1$ is established would lead to all keys in this session getting revealed, including the to-be-replaced key key_i. Thus, when $\mathcal{B}$ replaces the real key_i by a randomly chosen one, $\mathcal{A}$ is potentially able to determine this and abort, rendering our simulation invalid. This corresponds to the intuition that security cannot generally be expected for keys in a symmetric-key protocol that might be compromised through exposure of keys external to the symmetric-key security game.

Note that, moreover, we cannot get rid of Reveal queries for session keys $\mathsf{key}_{i'}$ of stages $i' \neq i$ in our simulation without sacrificing concurrent composition of the multi-stage key exchange protocol with several symmetric-key protocols at multiple stages. We can either implicitly obtain concurrent composition by allowing $\mathcal{A}$ to arbitrarily compromise session keys of stages $i' \neq i$ (which is what we do) or explicitly simulate the composed symmetric-key protocol on each stage ourself. In the latter case, though, in order to be able to correctly simulate the protocol for some stage, we would need to reveal the according session key ourself, i.e., issue exactly those Reveal queries that required key independence in the first place.

External usage. The composed key being used only externally of the key exchange protocol enables us to perform the hybrid steps via Test and argue that these do not affect the internal operations of the key exchange protocol. This corresponds to the intuition that a key which is already used within the key exchange protocol cannot at the same time be securely used in any arbitrary symmetric-key protocol. To give an example, say that the key is used in the key exchange to produce a MAC value, then composing the key with the according MAC protocol would—in the general sense—be insecure as an adversary can simply present the MAC value seen in the key exchange as a valid "fresh" forgery in the MAC security game.

Non-replayability. As for forward secrecy (and $\mathsf{Corrupt}$ queries), non-replayability ensures that $\mathsf{RevealSemiStaticKey}$ queries do not affect the simulation Π instances already spawned,

again capturing the intuition that such later compromise impede generic security of the composed protocol. Moreover, due to replays more than two sessions may derive the same session keys, a situation which contradicts the setting of two-party symmetric-key protocol security where keys are assumed to be shared between two parties only.

4.7 Further Work Extending the Model

We introduced the initial version of multi-stage key exchange model for the analyses of the QUIC protocol [FG14] and extended it along with the analysis of various TLS 1.3 drafts and handshake modes [DFGS15a, DFGS16, FG17] to the form presented here (cf. Chapters 5–7). Since then, and partially in parallel, the multi-stage key exchange model has been used and built upon by others in the following works.

Li et al. [LXZ+16] augment our multi-stage model by an additional notion of "levels" in order to analyze the cascading execution of multiple TLS 1.3 (resumption) handshakes as an alternative approach to the compositional security guarantees of our model (cf. Section 4.6).

Cohn-Gordon et al. [CGCD+17] build upon our multi-stage model to provide a formal analysis of the secure messaging protocol Signal [Sig]. They introduce a tweaked variant of the model to support a tree-like structure of stages instead of a linear sequence, in order to capture the complex key schedule and advanced security properties (like post-compromise security [CGCG16]) of the Signal protocol.

Brendel and Fischlin [BF17] adopt our multi-stage model in the smart card setting in order to analyze a 0-RTT extension of the Extended Access Control (EAC) protocol (see, e.g., [Int15]).

Cohn-Gordon et al. [CGCG+17] propose a design for a tree-based group key exchange protocol with strong security guarantees, in particular post-compromise security, to improve over current secure group messaging protocols. For their analysis, they define a group key exchange variant of our multi-stage model to capture their key updating mechanism.

Chapter 5

The QUIC Protocol

Summary. In this chapter we present our security analysis of the QUIC protocol proposed by Google in 2013 and globally deployed at Google in the meantime [QUI, LRW+17]. We first provide a cryptographic description of QUIC's key exchange protocol. We then give our security analysis and results in the multi-stage key exchange model and discuss its implications and notes on the protocol design. The results in this chapter are based on a work published at ACM CCS 2014 [FG14].

5.1 Introduction

In 2013, Google announced the QUIC [QUI] (for "Quick UDP Internet Connections") protocol, setting out to improve the efficiency of secure connections while maintaining security guarantees comparable to the de-facto standard TLS (Transport Layer Security) protocol. While, to this end, QUIC comprises substantial and complex engineering efforts to optimize efficiency, the focus in this chapter will be on the cryptographic connection establishment part of the protocol. QUIC employs a Diffie–Hellman-based key exchange protocol for establishing the keys for a secure connection, but particularly focuses on reducing the round complexity of the interactions. It starts with the client being able to deliver data to the server immediately—i.e., with zero round-trip time (0-RTT), protected under an intermediate cryptographic key. At some point, the server replies with its contribution to the key exchange. Both parties then switch to a stronger key and continue the interaction with that key. The basic version of the protocol is displayed in Figure 5.1.

As already outlined in Chapter 4, establishing multiple session keys (key_1 and key_2 in QUIC) demands a key exchange security model capturing interactions and dependencies between these keys. In QUIC, for example, we will see that the derivation of the final key key_2 being protected under the intermediate key key_1 makes the former's secrecy dependent on the latter's.

Analysis of QUIC. Google's QUIC protocol will hence be the first protocol we study in our multi-stage key exchange security model from Chapter 4. It provides a reasonable starting point as it is simpler than the TLS protocol, whose multi-stage security in version 1.3 we will turn to in Chapters 6 and 7. Investigating QUIC also avoids the need to deal with the problem of key deployment for the finished message as in TLS versions up to 1.2 [DR08], which lead researchers to use alternative approaches for security analyses [JKSS12, KPW13, BFS+13]. We show that QUIC is a secure key exchange protocol, assuming idealized key derivation via random oracles, the Gap Diffie-Hellman assumption [OP01], and use of a secure channel. Here we distinguish between the keys of the two stages, showing that the stage-one key provides basic key secrecy, whereas the stage-two key yields forward secrecy.

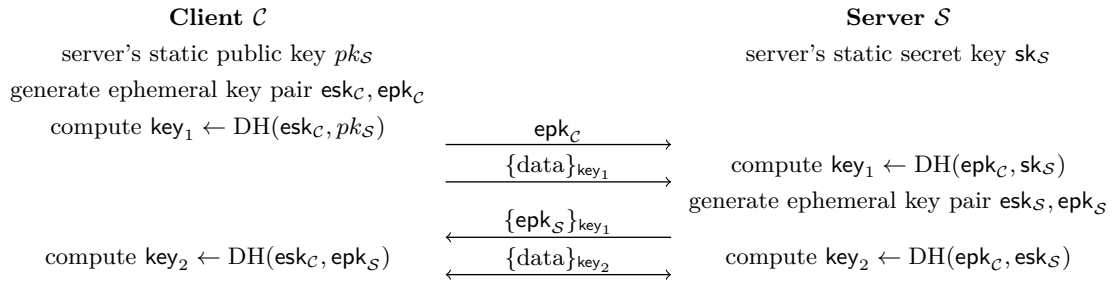


Figure 5.1: High-level protocol run description of Google's QUIC protocol with 0-RTT handshake. $\mathrm{DH}(X, Y)$ indicates the Diffie–Hellman secret computed from shares X and Y. $\{X\}_{\mathsf{key}}$ indicates sending X over a channel secured under key.

Note that our result about QUIC being a secure key exchange protocol shows that the protocol, as is, does not show any weakness, although the security bounds are far from being tight. Ideally, though, we would like to argue that QUIC, together with a secure channel protocol, provides a fully secure connection. This is where the compositional properties of our model and the composition result come into play. Recall from Section 4.6 that this result requires the key exchange protocol to be, in particular, (session-)key-independent and forward-secret. As the two keys derived in QUIC however are dependent, we first propose a slight modification of QUIC to turn it into a key-independent protocol, following ideas similar to those for TLS resumption. We then can conclude that compositional security with any symmetric-key protocol using the forward-secret second-stage session key is indeed achieved by the modified version of QUIC.

In summary, our results show that QUIC can be understood and analyzed as a multi-stage key exchange protocol. QUIC exhibits shows strong security properties, despite its comparatively low complexity. In particular, the trade-off between 0-RTT performance and forward secrecy is only one round trip. Still, as we discuss, with little effort QUIC can be strengthened further to facilitate a compositional analysis of its key exchange and channel component.

5.2 A QUIC Tour

Before we provide our analysis of the QUIC protocol, let us first give a more detailed description of QUIC and describe how we reflect the relevant protocol in our security model. Our discussion here and subsequent analysis refers to Revision 20130620 of the QUIC cryptographic protocol [LC13]. Most notably, this revision includes anti-replay mechanisms for the 0-RTT key established, which we discuss below and include in our analysis. In later revisions including the current Revision 20161206 [LC16], this replay protection mechanism was abandoned and the QUIC crypto protocol destined to be replaced by the upcoming TLS 1.3 protocol [LRW+17, TT17]. We discuss the different approaches towards replay protection for 0-RTT key exchange protocols in more detail in Chapter 7, but stress here that studying the QUIC design first is still meaningful, most importantly as it inspired the discussion and design of TLS 1.3's 0-RTT mode.

Let us recall the typical protocol run of the QUIC handshake shown in Figure 5.1 and in an expanded form in Figure 5.2. In the core protocol the client first sends an ephemeral Diffie–Hellman (DH) share from which the first session key key_1 is derived with the static DH share of the server, before the key key_2 of the second stage is derived when the server sends a temporary (semi-static) DH share over the key_1-secured channel (indicated by curly braces $\{\dots\}_{\mathsf{key}_1}$). The key key_2 is then computed as the DH key of the two ephemeral keys.

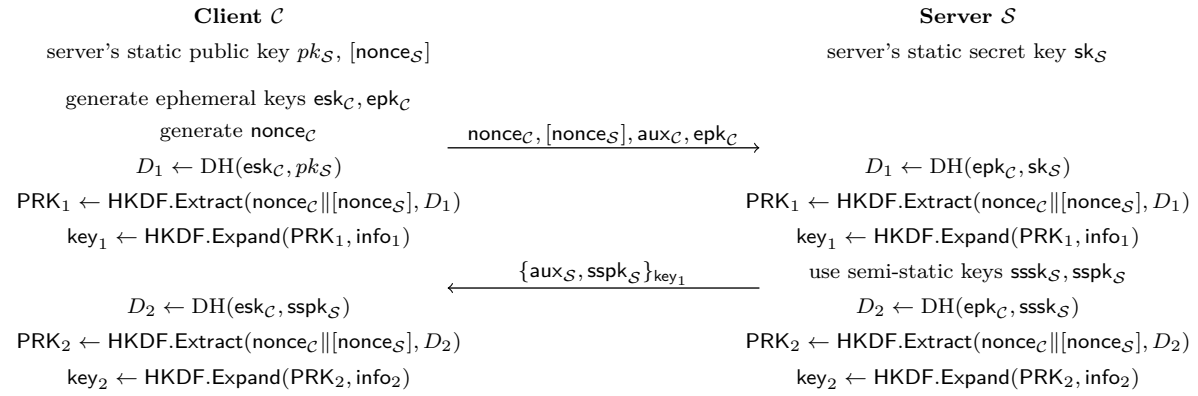


Figure 5.2: Expanded protocol run description of Google's QUIC protocol with 0-RTT handshake. $\mathrm{DH}(X, Y)$ indicates the Diffie–Hellman secret computed from shares X and Y. $\{X\}_{\mathsf{key}}$ indicates sending X over a channel secured under key.

Temporary keys. QUIC suggests to let the server use its Diffie–Hellman share in the second stage across multiple sessions to amortize the Diffie–Hellman generation at the server over the connections in that time span. The description [LC13] speaks of a life span of about 60 seconds in which the same share is used in every session of this server. These temporary keys are hence somewhat in between ephemeral per-session keys and static keys not bound to a single session. At the same time, they are too transient to be susceptible to cryptanalytic attacks, and QUIC relies on that as disclosure of the temporary keys indeed trivially exposes the derived (second-stage) key.

We therefore capture the temporary keys in QUIC as semi-static keys in our multi-stage key exchange model. This in particular allows the adversary in our model to decide when parties should switch to a new temporary key via a NewSemiStaticKey command, without introducing timing events. We however do not allow the adversary in our model to reveal these keys, i.e., we drop (access to) the RevealSemiStaticKey oracle.

Channels. Note that the key key_1 may be used to transmit payload data before it is used to establish key_2. In fact, the key_1-protected channel may still be used after the server has sent its share for key_2. The reason is the unreliable transmission due to QUIC running atop of UDP [Pos80], i.e., the ephemeral key may be delivered later than expected or even get lost. The actual channel protocol is not fully specified in the QUIC crypto specification [LC13], only references to possible authenticated encryption algorithms are given, supporting the usefulness of our composition theorem. We also remark that it turns out that for the security of the key exchange protocol we only need authenticity of the server's message, not confidentiality.

Certification. The main protocol is surrounded by some means to ensure that the server's static key pair is available and certified. Binding of keys to server identities is ensured by certification of public keys, potentially including revocation mechanisms. For the sake of simplicity, and in compliance with various similar efforts, we leave this part out of the security proof.[13] Hence, we presume that valid binding of static keys is ensured as a part of the security game in the sense that the assignment of public keys to parties is known by default.

[13]There are only a few exceptions where the certification process has been considered to be an integral part of cryptographic protocol, e.g. [BFPW07, FW09, BCF+13], where the latter one deals with key exchange explicitly.

If the client is currently not in possession of the server's public key it may start the interaction with an "inchoate" client hello. Upon receiving such a message, the server forwards its public configuration, possibly including the certificate and further information. We omit this part of the key retrieval in our modeling of the protocol, since we assume known binding of public keys to servers anyway.

Replay protection. To prevent replay attacks, QUIC employs the common countermeasure and uses nonces. However, because of the restriction of zero round-trip time, one cannot expect the server to contribute to the nonce, and must rely on the user to generate good nonces. To sustain security, QUIC assumes that the server uses a so-called "strike-register" in which previously seen nonces are stored. Several severs within a so-called "orbit" are supposed to share such a register. A nonce is thus assumed to consist of a time stamp, an orbit identifier, and 20 random bytes; the designers of QUIC estimate that 32 bytes should be sufficient.

If a connection with a client-generated nonce fails, because the server finds an entry in the strike register, then the server rejects, but provides a server-generated nonce, encrypted and authenticated under some private server key. If the server then recognizes such a server nonce in a subsequent, fresh 0-RTT connection retry, it can check that it is authentic. We simply write $\mathsf{nonce}_{\mathcal{C}}$ for the nonce eventually used by the client, and $[\mathsf{nonce}_{\mathcal{S}}]$ for the (optional, otherwise empty) server nonce. Because of the strike registers, we presume in our protocol abstraction that any honest server accepts any client nonce only once.

We note that there are practical scenarios, particularly in distributed server settings, where replay protection on the key exchange level (as provided by QUIC in Revision 20130620) cannot successfully prevent replays on the application level. We provide a detailed discussion of such attacks and (the limits of) mitigation strategies in Chapter 7.

Format of handshake messages. Handshake messages are tagged, e.g., the client resp. server hello message in the handshake phase carry special tags CHLO and SHLO, and may contain further information like the version numbers. However, many of these entries are optional and do not directly contribute to the cryptographic strength of the key exchange step (except that they enter the key derivation step in a non-critical way, see below). We thus simply write aux for these data, with a subscript for the corresponding party.

Key derivation. Key derivation is performed via the key derivation function HKDF [Kra10, KE10] based on HMAC [BCK96, KBC97] (cf. Section 3.2.3) with hash function SHA-256, as specified by NIST SP800-56C [Che11]. This is a two-stage derivation process. In the first extraction step via function HKDF.Extract one computes a pseudorandom master key PRK from the corresponding Diffie-Hellman key, using the client nonce and possibly the server nonce as a salt input. In our security proof we model this extraction function as a random oracle.

In the second expansion step, one derives client and server write keys and IV values by expanding the PRK via HKDF.Expand. Here, the input are the client hello message, the (public) server configuration, and a label distinguishing the first-stage key ("QUIC key expansion") from the second-stage key ("QUIC forward secure key expansion"). We denote these data by info_1 and info_2, respectively. Note that they are both determined given the client's ephemeral key, the nonces, the auxiliary data and the stage number. We assume in our analysis that HKDF.Expand is a random oracle, too.

Session identifiers and partners. For the analysis we also need to specify the intended partners and the session identifiers. Since clients are not authenticated in QUIC, we assume that the responder in an execution, i.e., the server, sets the partner identity label.pid to '$*$'. The

client on the other hand sets the partner entry to the identity of the server specified through the public key. As for session identifiers, for both parties we let $\mathsf{sid}_1 = \mathsf{info}_1 = (pk_{\mathcal{S}}, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{aux}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}})$ and $\mathsf{sid}_2 = (\mathsf{info}_1, \{\mathsf{aux}_{\mathcal{S}}, \mathsf{sspk}_{\mathcal{S}}\}_{\mathsf{key}_1})$, where the latter value is the authenticated ciphertext sent by the server. Note that the session identifiers are only set to these values once the corresponding party accepts, and are $\perp$ otherwise. Each stage consisting of a single message only, the contributive identifiers are set identically to the session identifiers for each stage. We remark that $\mathsf{aux}_{\mathcal{C}}$, containing the used server configuration's ID, together with the verified certification of the server configuration uniquely identifies the full configuration used in the key derivation. Furthermore, observe that info_1 and info_2 can be derived mutually from another, as they only differ in some constant labels.

5.3 Security of QUIC

We conduct our security analysis of QUIC in the (public-key) multi-stage key exchange (MSKE) model (cf. Chapter 4). In terms of this model, we will establish that QUIC is key-dependent and stage-2-forward-secret with the following protocol-specific properties $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$:

- $\mathsf{M} = 2$: QUIC consists of two stages deriving keys key_1 and key_2.
- $\mathsf{AUTH} = \{(\mathsf{unilateral}, \mathsf{unilateral})\}$: QUIC fixes unilateral (responder) authentication for both stages.
- $\mathsf{USE} = (\mathsf{internal}, \mathsf{external})$: The first key established in QUIC is used within the key exchange (to encrypt the server's response), the second key is only used externally.
- $\mathsf{REPLAY} = (\mathsf{nonreplayable}, \mathsf{nonreplayable})$: Both stages' keys are non-replayable due to the strike register employed in the protocol.

As discussed above, we will omit the RevealSemiStaticKey oracle from the model, as QUIC does not aim at security when the re-used temporary server Diffie–Hellman share is exposed.

For the security proof we will rely on the random oracle model and the Gap-Diffie–Hellman (GapDH) problem [OP01] (cf. Section 3.2.1).[14] Besides the underlying number-theoretic problem, we also need security of the channel protocol which is used for the server hello message. Since we only need authenticity, we can simply define security as follows: We denote by $\mathsf{Adv}^{\mathrm{Auth}}_{\{\cdot\},\mathcal{A}}$ the probability that adversary $\mathcal{A}$, when allowed to query the channel oracle $\{\cdot\}_{\mathsf{key}}$ for a random key key at most once, is able to create in an attempt a channel message (not returned by the oracle) such that decryption under key yields a valid message. Note that we merely require one-time authenticity because we analyze QUIC as a key exchange protocol only, assuming that no payload data is sent by the client in the first stage.

Theorem 5.1 (Match security of QUIC)**.** *QUIC is* Match*-secure with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. For any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{QUIC,\mathcal{A}} \leq n_s^2/q,$$

where n_s *is the maximal number of initiated sessions and* q *denotes the size of the group* $\mathbb{G}$*.*

[14] We note that alternatively to using GapDH in the random oracle model we could employ the (multiple-single) double-sided msPRF-ODH variant of the pseudorandom-function oracle-Diffie–Hellman assumption (instantiable under the same assumptions [BFGJ17], cf. Section 3.2.2). This would enable the necessary multiple queries to the challenged server's static secret in other sessions as well as a potential single query to the challenged client's ephemeral secret for a differing server ephemeral secret, similar to the usage of msPRF-ODH in the proof for the TLS 1.3 `draft-12` (EC)DHE 0-RTT handshake in Section 7.5. We here stick to the original proof from [FG14], in particular enabling a comparison of both proof techniques.

Proof of Theorem 5.1. We need to show the six properties of Match security (cf. Definition 4.1).

1. *Sessions with the same session identifier for some stage hold the same key at that stage.*
   Note that identical session identifiers in QUIC (at either stage) imply that the input to the key derivation functions are identical, too. Hence there cannot exist stages with identical session identifiers but different keys.

2. *Sessions with the same session identifier for some stage agree on that stage's authentication level.*
   This trivially holds as QUIC fixes unilateral (responder) authentication for all stages.

3. *Sessions with the same session identifier for some stage share the same contributive identifier.*
   This immediately follows from the session identifiers being equal to the contributive identifiers in both stages.

4. *Sessions are partnered with the intended (authenticated) participant.*
   The client does not authenticate. The server's public key in QUIC is part of both session identifiers and that the identity can be reliably deduced from the key resp. the certificate by assumption, thus the property holds.

5. *Session identifiers are distinct for different stages.*
   This is immediately satisfied by sid_2 containing more elements than sid_1.

6. *At most two sessions have the same session identifier at any non-replayable stage.*
   Note that the probability that two sessions of honest clients create the same random ephemeral key is at most n_s^2/q by the birthday bound. Here we use that corrupting a user terminates the interaction with the session such that, in particular, that session does not generate session identifiers. Given that no such collision occurs, the three sessions in question must include two sessions of honest servers. But the client nonce, appearing both in sid_1 and in sid_2, contains the server's orbit and this value also enters the session identifier. Hence, the two servers in the same orbit must have accepted the same client nonce twice, contradicting our assumption about the strike registers. □

Theorem 5.2 (Multi-Stage security of QUIC)**.** *In the random oracle model, QUIC is* Multi-Stage*-secure in a key-dependent and stage-2-forward-secret manner with properties* (M, AUTH, USE, REPLAY) *given above. For any efficient adversary $\mathcal{A}$ there exist efficient algorithms $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage}}_{QUIC,\mathcal{A}} \leq 2n_s \cdot \Big((n_s n_u + n_s n_{ss}) \cdot \mathsf{Adv}^{\mathsf{GapDH}}_{\mathbb{G},\mathcal{B}} + (2q_h + 4q_h n_s) \cdot 2^{-\min\{|\mathsf{PRK}_1|,|\mathsf{PRK}_2|\}} + n_s \cdot \mathsf{Adv}^{auth}_{\{\cdot\},\mathcal{C}} \Big),$$

where n_s is the maximal number of sessions, n_u is the maximal number of users, n_{ss} is the maximal number of semi-static keys generated, q_h is the total number of random oracle queries of the adversary, and $|\mathsf{PRK}_i|$ is the output length of HKDF.Extract*.*

Proof of Theorem 5.2. First, we may consider the case that the adversary makes a single Test query only. This can decrease the success probability by a factor at most $1/2n_s$ by a hybrid argument (leveraging that session matching is decidable based on the public transcripts) as there are at most $2n_s$ keys. From now on we can therefore speak of *the* tested session. Recall further that for an admissible Test query in a responder-authenticated unilateral protocol, the query must be either for an initiator session (i.e., for a client in QUIC), or for a partnered server session such that the client's ephemeral public key originates from a session of an honest client.

Stage-1 secrecy. Consider first the (non-forward) secrecy of the session keys of the first stage. We can bound the adversary's success probability to distinguish the keys from random by (a) the probability that the adversary queries the random oracle HKDF.Extract about the DH key (specified through the session identifier of the tested session), plus (b) the conditional probability that $\mathcal{A}$ succeeds given that it has not queried HKDF.Extract about the key before. In the latter case, the corresponding value PRK_1 is an unknown random value for the adversary. Furthermore, since the adversary cannot reveal the session key in partnered sessions and keys for other session identifiers are distributed independently, distinguishing the derived test session key from random is then given by the (pseudo)randomness of HKDF.Expand. To be precise, we can bound the adversary's advantage by its number of queries to the random oracle in proportion to the size of possible PRK_1 values, i.e., by $q_h \cdot 2^{-|\mathsf{PRK}_1|}$.

The former probability of making the query to HKDF.Extract about the DH value can be bounded in terms of the GapDH assumption, along the arguments for similar protocols, e.g. [JP02, KP05, LM06, DF11]. That is, one guesses two sessions, one being a client session, the other one being a server session, and injects the given challenge values X and Y of the GapDH problem into the client's ephemeral key and the server's static public key. The hope is that these sessions will correspond to the Test query, which is either for a client session, or for a server session, but which is then partnered to the (hopefully correctly predicted) client session and key. If the adversary makes the random oracle query about the DH key of the two values, then we can solve the DH problem. Here, in the course of the simulation, the server's long-term key may be used in another session, in which we could not derive the corresponding DH key. Using the same technique as in previous works, we leverage the decisional DH oracle to simulate the random oracle via implicit representation of DH tuples.

More formally, we build a reduction $\mathcal{B}$ to the GapDH problem as follows. We are given $\mathbb{G}$, g and two random group elements X, Y and are supposed to compute $Z = \mathrm{DH}(X, Y)$ with the help of a decisional oracle $\mathrm{DDH}(\cdot, \cdot, \cdot)$. We initially guess one of the at most n_s executions and one of the at most n_u server keys at random. We will use X in the predicted execution as the honest client's ephemeral key (and abort if the session starts but is not by a client nor by an honest party), and analogously use Y as the server's long-term public key. Run now the attack of the stage-1 adversary by emulating the honest parties' behavior, with the only exception that honest parties sometimes need to skip hash computation and instead maintain an implicit representation. This is necessary in the case that the injected keys appear and we cannot compute the DH values on behalf of the honest parties. We will match this list against the explicitly computed hash values by the adversary. Note that the adversary will be oblivious about this structural modification, as we still simulate the random oracle as before and the input/output behavior of the honest parties are statistically indistinguishable from its point of view.

To simulate the execution we maintain an initially empty list and use it as follows to compute hash answers for both stages:

- If the adversary makes a hash query to the extraction random oracle HKDF.Extract about $(\mathsf{nonce}_\mathcal{C} \| [\mathsf{nonce}_\mathcal{S}], D)$, then we return a (consistent) random answer PRK via lazy sampling, i.e., where we answer previous queries as before and pick a fresh value for a new query. Next we check if we can update our list by searching for entries $(\{A, B\}, \mathsf{nonce}_\mathcal{C}, [\mathsf{nonce}_\mathcal{S}], \mathsf{info}, *, \mathsf{key})$ with $\mathrm{DDH}(A, B, D) = 1$ and where the value for PRK has not been set yet.[15] Note that we can check for this efficiently since the size of the list will be bounded by the number of sessions, and each element can be checked easily

[15] Here, and also below, the optional server nonce $[\mathsf{nonce}_\mathcal{S}]$ should only be used in the list operations if it also appears in the hash query, e.g., if the adversary queries about $(D, \mathsf{nonce}_\mathcal{C})$ then we also search the list for entries $(\{A, B\}, \mathsf{nonce}_\mathcal{C}, \mathsf{info}, *, \mathsf{key})$.

with the help of the decision oracle. If we find such an entry then we set the wildcard $*$ to PRK.

- If the adversary makes a query $(\mathsf{PRK}, \mathsf{info})$ to the expansion random oracle $\mathsf{HKDF.Expand}$ we first search for entries $(\{A, B\}, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{info}, \mathsf{PRK}, \mathsf{key})$ with matching entries for info and PRK in our list. If we find such an entry then we return key. Else we answer (consistently) as the random oracle would.

- If a (simulated) honest party is supposed to compute a key for group elements A and B, nonces $\mathsf{nonce}_{\mathcal{C}}$ and $[\mathsf{nonce}_{\mathcal{S}}]$, and execution information info, then we proceed as follows: If the party could compute the DH key D itself we do so and proceed as in the adversarial cases above, possibly updating information in our list. If the party could not compute the DH key, say, because it involves the injected server's long-term key Y, then it searches for an entry $(\{A, B\}, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{info}, \mathsf{PRK}|*, \mathsf{key})$ in the list (where '$\mathsf{PRK}|*$' stands for 'either PRK or $*$') and subsequently uses key. If there is no such entry then it picks key at random for subsequent usage, and adds an entry $(\{A, B\}, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{info}, *, \mathsf{key})$ to the list.

The list strategy basically allows the reduction to implicitly set the PRK value and adjust it later. An inconsistency can happen if the adversary asks the expansion oracle $\mathsf{HKDF.Expand}$ about a value $(\mathsf{PRK}, \mathsf{info})$ to receive a key key, before having received PRK as a reply from the extraction oracle $\mathsf{HKDF.Extract}$. If we later set the wildcard $*$ in our list to that value PRK but for a different key, then this does not match the adversary's expectation. However, since the value PRK is chosen at random, the probability that this happens among the at most q_h random oracle queries of the adversary and the at most $2n_s$ list entries (of both stages) is at most $2q_h \cdot n_s \cdot 2^{-\min\{|\mathsf{PRK}_1|, |\mathsf{PRK}_2|\}}$.

Recall that we assume that the adversary makes a hash query to derive PRK_1 in the Test session. We can check for all queries via the decisional oracle if this has already happened; if so we can output the correct value and solve the GapDH problem in this case. Also observe that the Test session must be either between an honest client and an uncorrupted server, or that the server must be honest and the client's ephemeral must origin from an honest client. Therefore, given that the simulation does not generate any inconsistency, our simulation is perfectly indistinguishable from an actual attack of the adversary's viewpoint. In particular, the Test session then uses our injected keys X, Y with probability at least $\frac{1}{n_s \cdot n_u}$, allowing us to solve the GapDH problem in this case.

Finally, to complete the argument, note that the adversary cannot succeed by hoping that another session with different session identifier sid_1' yields the same input $(\mathsf{PRK}_1, \mathsf{info}_1)$ to $\mathsf{HKDF.Expand}$. This would potentially allow the adversary to Reveal that session key and distinguish the tested key from random. The reason is simply that the session identifier information completely enters the key derivation and the session keys of distinct sessions are thus distributed independently.[16]

Stage-2 forward secrecy. To show stage-2 forward secrecy, we distinguish again between the cases that the adversary queries the random oracle about the DH key of stage 2, and that it does not (in which case the randomness of PRK_2 ensures security of the session keys again with a bound of $q_h \cdot 2^{-|\mathsf{PRK}_2|}$). For the first case, however, we have to apply a more fine-grained case distinction now. To this end, we first show that the adversary essentially cannot inject its own temporary/semi-static key into the server's hello message; else this would clearly violate security. For this we argue that the first stage key key_1 of the tested client session with label

[16] The derived keys may be identical by chance but this does not violate our analysis.

label still looked random to the adversary when the server hello message has been sent but not yet received. This follows as above and from the following three properties:

1. Because of the key dependence, the adversary cannot learn the key key_1 via a $\mathsf{Reveal}(\mathsf{label}, 1)$ query to the test session; such queries are prohibited before the key key_2 has been established.

2. For the same reason, key dependence, the adversary cannot learn key_1 by revealing the key of a session label' which is partnered according to the stage-one session identifier ($\mathsf{label}.\mathsf{sid}_1 = \mathsf{label}'.\mathsf{sid}_1$). Any such reveal request would make key_1 and key_2 in the tested session revealed, according to the Reveal query in which keys for partnered sessions are set to revealed for the current and all subsequent stages.

3. Corruptions of the test session's party could only have happened after key_2 has been established.

Since key_1 has looked fresh, we can then argue along the authenticity of the key_1-channel. The adversary either gets to see one or none channel message for the fresh key key_1 (depending on whether there is a partnered session to label), and needs to break the authenticity if it manages to send a new valid ciphertext. This is bounded by advantage $\mathsf{Adv}^{\mathrm{auth}}_{\{\cdot\},\mathcal{C}}$ times the factor to guess the right sessions again.

More formally, we consider the probability that the adversary in the attack sends to the (honest) client in an execution for info_1 a ciphertext which the client does not reject but which has not been created by the (honest) server, as specified in info_1. Note that this comprises the case that the adversary tries to forge an authentic ciphertext from scratch, or that it has forwarded the client's first message to the server and got a different, valid ciphertext as reply. We bound this probability by the advantage of an adversary against the authenticity of the channel protocol.

Our adversary $\mathcal{C}$ against the channel basically simulates the honest parties for the key exchange attacker with one exception: It initially selects one of the at most n_s client sessions (with identifier info_1) at random and waits for an honest server session in which the client has sent $\mathsf{nonce}_{\mathcal{C}}, \mathsf{aux}_{\mathcal{C}}$, and $\mathsf{epk}_{\mathcal{C}}$ of info_1 and is supposed to answer using the key key_1. Note that by the strike registers this server session is uniquely determined. Our adversary prepares the server's answer according to the protocol, e.g., using its current semi-static key, but then eventually calls its external channel oracle to create the authenticated ciphertext (under a fresh key). If the adversary against the key exchange protocol sends a reply to the predicted client session involving the same data info_1, then we output this ciphertext as a potential forgery.

By the argument for stage-1 security, and the fact that the adversary cannot learn key key_1 in session info_1 by other means like Reveal queries because of key dependence, it follows that using the fresh key instead of key_1 cannot influence the adversary's success probability significantly. Intuitively, one may think of key_1 as having been replaced by the random value used in the game $\mathsf{Adv}^{\mathrm{Auth}}_{\{\cdot\},\mathcal{C}}$. Hence, a key-exchange adversary as above would essentially win with the same probability as in game $\mathsf{Adv}^{\mathrm{Auth}}_{\{\cdot\},\mathcal{C}}$, times the probability n_s for predicting the client session.

We conclude that we can from now on reject any attempt in which the key-exchange adversary sends to an honest client a new ciphertext which has not been created by an honest server. The adversary can thus only relay the second messages between an honest client and an honest server. In such an execution we can again inject the GapDH challenge X, Y into the client's ephemeral public key and the server's semi-static public key. Only this time, we have to guess one session and the right semi-static key used by the server, instead of one session and the right (long-term secret of a) user, yielding a factor $n_s \cdot n_{ss}$ instead of $n_s \cdot n_u$. Note that $\mathsf{Corrupt}$ queries

for the server only disclose the long-term secret, but not the semi-static key by convention, and we do not allow the adversary to access the RevealSemiStaticKey oracle here. Hence, we can carry out the same reduction to the GapDH problem as above.

The final step, as in the stage-1 case, is now to argue that there cannot be another session with identifier $\mathsf{sid}_2' = (\mathsf{info}_1', C') \neq \mathsf{sid}_2 = (\mathsf{info}_1, C)$ such that the inputs to the derivation function HKDF.Expand are identical. Recall that each info_1 contains the client's ephemeral public key and that it corresponds uniquely to some info_2. Hence, a difference in $\mathsf{info}_1' \neq \mathsf{info}_1$ would immediately yield different inputs $\mathsf{info}_2' \neq \mathsf{info}_2$ to HKDF.Expand in the protocol. If, on the other hand, $\mathsf{info}_1' = \mathsf{info}_1$ then the two ciphertexts must differ. Because of the strike registers on the server's side the ciphertexts can only differ if one has been created by the adversary for the same stage-1 key. In this case, however, we would have rightfully rejected the ciphertext such that the client would not have derived the session key key_2. It follows that only the partnered sessions can have the same input $\mathrm{DH}(\mathsf{epk}_{\mathcal{C}}, \mathsf{sspk}_{\mathcal{S}})$ and info_2 to HKDF.Expand, implying that the key in the test session is independent of all other keys (except for the keys of partnered sessions).

The claim now follows. □

Making QUIC key-independent. Recall that our composition theorem in particular only applies to key-independent protocols, whereas QUIC, as is, does not satisfy this property. It is, however, quite easy to change QUIC into a key-independent version. With this modification we can then argue security of, say, the composition with a secure channel protocol for the second stage.

Recall that, in the key-independent case, the adversary is allowed to reveal the session key of a stage before the session key of the next stage has been established. The idea for QUIC is similar to, e.g., TLS resumption, where the resumption key is derived from an established master secret (from which the previous session keys have been computed). For a key-independent variant of QUIC, one would simply derive two secret values key_1 and preK_2 in the HKDF.Expand step of the key derivation in the first stage, where key_1 is still the first stage's session key and preK_2 is kept secret and subsequently input to the key derivation in the second stage. Any Reveal query would then disclose the session keys, but not preK_2. It should thus be hard to compute the second-stage session keys given only the previous session keys.[17] We stress that this change does not impose additional expensive state to be kept by the server: As apparent from Figure 5.2, the server computes key_2 immediately after deriving key_1 and must anyway keep a small state between the two KDF invocations.

[17] Note that, if we would allow session state reveals, then the key PRK could still be disclosed, of course. The idea here therefore protects against bad usage of the session keys in the channel (modeled through Reveal queries), but not against disclosure of ephemeral randomness.

Chapter 6

The TLS 1.3 Protocol: Diffie–Hellman and Pre-shared Keys

Summary. In this chapter we present our security analysis of the TLS 1.3 handshake protocol candidates in version `draft-10` [Res15e] for the main Diffie–Hellman-based handshake mode and the abbreviated resumption mode based on pre-shared keys. We first describe the cryptographic operations of both modes and then give our security results in the multi-stage key exchange model. We also discuss some of the design choices in TLS 1.3 along the way. The results in this chapter are based on a work published ACM CCS 2015 [DFGS15a, DFGS15b] and one presented at the TLS 1.3 TRON ("TLS 1.3 – Ready or Not?") workshop 2016 [DFGS16].

6.1 Introduction

The *Transport Layer Security (TLS)* protocol [DR08] is one of the most widely deployed cryptographic protocols in practice, comprising both key exchange (the *handshake protocol*) and secure channel (the *record protocol*) components. Due to numerous security problems with the existing versions of TLS up to the latest one, TLS 1.2 [DR08], but also because of additional desirable privacy features (both discussed in Section 1.1.3), the Internet Engineering Task Force (IETF) is currently drafting a *new TLS 1.3 standard*.

Work on TLS 1.3 began in mid 2014, when the first draft versions were defined based on the previous TLS 1.2 version [DR08]. We conducted our first analysis of TLS 1.3 in May 2015 [DFGS15a, DFGS15b]; at this point there were two (slightly different) candidates in discussion: one is `draft-05` [Res15b], the other one is the forked `draft-dh` [Res15g], incorporating a different key schedule based on the OPTLS protocol design by Krawczyk and Wee [KW15, KW16]. A later analysis of ours [DFGS16] provided an updated analysis of the design of `draft-10` [Res15e], published in October 2015. Both our analyses of `draft-05` and `draft-dh` [DFGS15a, DFGS15b] as well as `draft-10` [DFGS16] provide a comprehensive cryptographic evaluation of the primary Diffie–Hellman-based handshake as well as the (resumption) handshake based on pre-shared keys (PSKs), to the extent specified, of the respective drafts. In a later work [FG17] we then also conducted an analysis of both the Diffie–Hellman-based and the PSK-based 0-RTT handshake modes specified in `draft-12` [Res16a] resp. `draft-14` [Res16c]. The most recent TLS 1.3 draft is `draft-24` [Res18] which differs from the latest drafts we analyzed in some cryptographic aspects, particularly details in the key schedule, but overall still defines the handshake modes similar to the drafts covered by our analysis.

In this chapter, we cover the results for the Diffie–Hellman-based and PSK handshakes of `draft-10`, providing comparative remarks to other draft versions along the way. Chapter 7

then contains our security analyses of the (DH- and PSK-based) 0-RTT handshake modes from `draft-12` and `draft-14`, particularly focusing on replays in this setting. While the results presented in this thesis hence do not capture the changes made in the latest TLS 1.3 drafts, they provide a thorough evaluation of the core cryptographic design of all TLS 1.3 handshake modes and give an insight into the design process and decisions for this new version of TLS. We particularly believe that it is important that cryptographic evaluation takes place *before* standardization. This contrasts with the history of TLS and its predecessor the Secure Sockets Layer (SSL) protocol (see also Paterson and van der Merwe [PvdM16]): SSL 3 [FKK11] was standardized in 1996, TLS 1.0 [DA99] in 1999, TLS 1.1 [DR06] in 2006, and TLS 1.2 [DR08] in 2008, but the first comprehensive cryptographic proof of any complete TLS ciphersuite did not appear until 2012 [JKSS12]. We refer to Section 1.3 for an extensive discussion of the prior and related work on (analyses of) the TLS protocol.

The TLS 1.3 handshake protocol design introduces several cryptographic changes that are substantially different from TLS 1.2, including: (1) encrypting some handshake messages with an intermediate session key, to provide confidentiality of handshake data such as the client certificate; (2) signing the entire handshake transcript for authentication; (3) including hashes of handshake messages in a variety of key calculations; (4) encrypting the final `Finished` messages in the handshake with a different key than is used for encrypting application data; (5) deprecating a variety of cryptographic algorithms (including RSA key transport, finite-field Diffie–Hellman key exchange, SHA-1, RC4, CBC mode, MAC-then-encode-then-encrypt); (6) using modern authenticated encryption with associated data (AEAD) schemes for symmetric encryption; (7) updating keys during execution of the record protocol to increase the channel's lifetime and security; and (8) providing handshakes with fewer message flows to reduce latency.

These changes are meant in part to address several of the aforementioned attacks. While some of those attacks are implementation-specific and escape abstract cryptographic evaluation, assessing the cryptographic security of the design of TLS 1.3[18] can provide assurance that the protocol design does not display any unexpected cryptographic weaknesses. Our goal is a comprehensive assessment of the security of the handshake protocol, covering `draft-10` in this chapter. Overall, in the first part of this thesis we focus solely on the handshake protocol as a key exchange protocol (aspects of the TLS 1.3 channel are captured in Part II). Our modular approach to treat both components individually is made possible by the TLS 1.3 drafts providing a cleaner separation between the key exchange in the handshake protocol and the use of the resulting session key in the record protocol. This contrasts with TLS 1.2 and earlier, where the session key was used both for record layer encryption and encryption of the `Finished` messages in the handshake, making it impossible for TLS 1.2 to satisfy standard key exchange indistinguishability notions and requiring either (a) a more complex security model that treats the handshake and record layer together [JKSS12] or (b) a cunning approach to release the record layer key early [BFK+14]. The cleaner separation in the TLS 1.3 design allows us to take a compositional approach to the security of TLS 1.3, treating the handshake separate from the record layer, and also allowing us to include session resumption for abbreviated handshakes.

6.1.1 Modeling TLS 1.3 as a Multi-Stage Key Exchange Protocol

The message flow for the TLS 1.3 `draft-10` full, i.e., (elliptic-curve) ephemeral Diffie–Hellman (abbreviated (EC)DHE) handshake is shown in Figure 6.1 (on page 61) along with the respective key schedule. The `draft-10` PSK-only and the combined PSK with ephemeral Diffie–Hellman handshakes (abbreviated PSK resp. PSK-(EC)DHE) are given in Figure 6.2 (on page 74). It is

[18] When we refer to "TLS 1.3", we mean the common features of its design conceptually spanning all drafts discussed here.

convenient to view the TLS 1.3 handshakes as *multi-stage* key exchange protocols in which both parties, the client and the server, agree on multiple session keys, with potential dependencies between these keys.

In the first stage, a handshake traffic key tk_{hs} is derived. In the full or PSK-(EC)DHE handshake, tk_{hs} is established via an anonymous Diffie–Hellman key exchange (in the `ClientKeyShare` and `ServerKeyShare` messages). In the PSK-only handshake, tk_{hs} is computed from the shared pre-shared key PSK. From the established shared Diffie–Hellman value resp. the shared PSK, denoted as ephemeral secret ES, both parties first compute an (HKDF-)extracted version xES. This value is then used to compute tk_{hs} which encrypts the remaining messages of the handshake and should provide some form of outsider privacy, e.g., for the exchanged certificates.

In the second stage, the parties in the full handshake (depending on the desired authentication level) exchange signatures over the (hash of the) transcript under a certified key in order to authenticate. Both parties in all handshake modes conclude the protocol by exchanging `Finished` messages over the transcripts, generated a distinct finished secret key FS. Via further key derivation steps, both parties then compute a master secret MS (mixing in the the pre-shared key in PSK-based modes). From that, they then derive the application traffic key tk_{app} for securing the actual application data. They furthermore derive from MS a resumption master secret RMS (in the full handshake) for potential session resumption as well as an exporter master secret EMS which can be used for deriving additional keying material. Viewing each of these keys as one of the (three resp. four) multi-stage session keys enables us to argue about their security, even if the other keys are leaked.

We present the full and PSK-based handshake modes in more detail in Sections 6.2 and 6.4. The security results using our multi-stage key exchange model follow in Sections 6.3 and 6.5, respectively.

Security of the `draft-10` full (EC)DHE handshake. We show that the full (EC)DHE handshake of `draft-10` is a secure multi-stage key exchange protocol where different stages and simultaneous runs of the protocols can be unauthenticated, unilaterally authenticated, or mutually authenticated. On a high level, this means that the handshake establishes record layer as well as resumption and exporter keys that look random to an adversary. This holds even for sessions that run concurrently and if the adversary controls the network, is able to corrupt the long-term secret keys of other parties, and allowed to reveal keys established in other sessions, thus providing quite strong security guarantees for practice. Moreover, the multi-stage model allows us to show that even leakage of record layer or exporter keys in the same handshake session do not compromise each other's security. All keys derived in the `draft-10` full (EC)DHE handshake enjoy forward secrecy and key independence, making them in particular amenable to a compositional analysis approach of their usage in the record protocol.

Notably, we are able to prove a standard (Bellare–Rogaway-style) notion of key secrecy (or key indistinguishability) for the handshake as key exchange protocol, which is in contrast to the results possible for previous TLS versions [MSW08, GMSS08, JKSS12, BFK+14]. Our security proof relies on mostly standard cryptographic assumptions such as unforgeability of the deployed signature scheme, collision resistance of the hash function, and pseudorandomness of the HKDF key derivation function. In addition, we employ the pseudorandom oracle-Diffie–Hellman (PRF-ODH) assumption in the snPRF-ODH variant which has been introduced and used for analyses of the previous TLS version 1.2 [JKSS12, KPW13].[19] See Section 3.2 for the

[19] In a recent work [BFGJ17], we found strong indications that the PRF-ODH assumption likely cannot—as originally hoped for [JKSS12, KPW13]—be instantiated in the standard model. We still deploy it in our TLS 1.3 analyses as a convenient stepping stone in the proof to avoid a tailored and involved reduction to the Strong or Gap Diffie–Hellman assumption in the random oracle (cf. the security proof for QUIC in Section 5.3 for a comparison).

definitions of these assumptions.

Security of the `draft-10` PSK and PSK-(EC)DHE handshakes. We also analyze the pre-shared key handshake modes of `draft-10`, PSK and PSK-(EC)DHE, and show that they as well are secure multi-stage (pre-shared secret) key exchange protocols. The result holds under similar cryptographic assumptions, particularly relying on the unforgeability of the HMAC message authentication code instead of signature unforgeability for authentication. The two pre-shared key modes differ in that the plain PSK handshake does not achieve forward secrecy while the PSK-DHE handshake, mixing fresh ephemeral Diffie–Hellman keys into the key derivation, does indeed establish forward-secret keys as envisioned.

Composition with the record protocol. When it comes to the overall security of TLS 1.3, we follow a compositional approach. More specifically, we leverage our generic composition theorem for the multi-stage key exchange model (see Section 4.6) to establish that all forward-secret external keys (i.e., those not used in the handshake itself) established can safely be used in any subsequent symmetric-key protocol. We can deduce such guarantees for the application traffic key tk_{app} as well as the resumption and exporter master secrets RMS resp. EMS of the full and PSK-(EC)DHE handshakes.[20] This means that, in particular, the cascading usage of the resumption master secret of the full handshake as an input to later PSK/PSK-DHE handshake runs is safe. Likewise, it allows an independent analysis of the record layer using the application traffic key; e.g., by applying the security models and results for channels presented in Part II of this thesis.

6.2 The TLS 1.3 draft-10 Full (EC)DHE Handshake Protocol

The TLS 1.3 full (EC)DHE handshake protocol can be conceptually subdivided into three phases:

Key exchange. In the key exchange phase, parties negotiate the ciphersuites and key-exchange parameters to be used and establish shared key material as well as traffic keys to encrypt the remaining handshake.

Server parameters. In the server parameters phase, further handshake parameters (as, e.g., whether client authentication is demanded) are fixed by the server.

Authentication. In the authentication phase, both the server and client can (based on the aspired authentication) authenticate, verify that they share the same view of the handshake, and derive (authenticated) application traffic keys as well as resumption and exporter keys.

Figure 6.1 shows the message flow and relevant cryptographic computations as well as the key schedule for the full (EC)DHE handshake in `draft-10`. The handshake messages are the following:

- `ClientHello` (`CH`)/`ServerHello` (`SH`) contain the supported versions and ciphersuites for negotiation purposes, as well as random nonces r_c resp. r_s. Both `CH` and `SH` can also

[20] In our original analysis of `draft-10` we debated an additional `NewSessionTicket` used for session resumption and sent encrypted under tk_{app} after the full handshake impaired the compositional guarantees for that key. In the light of follow-up revisions of TLS 1.3, we here consider `NewSessionTicket` as one of several so-called "post-handshake messages" (cf. Section 6.2) which can be sent at an arbitrary later point in time after the full handshake, and hence do not treat it as part of the main handshake we analyze here.

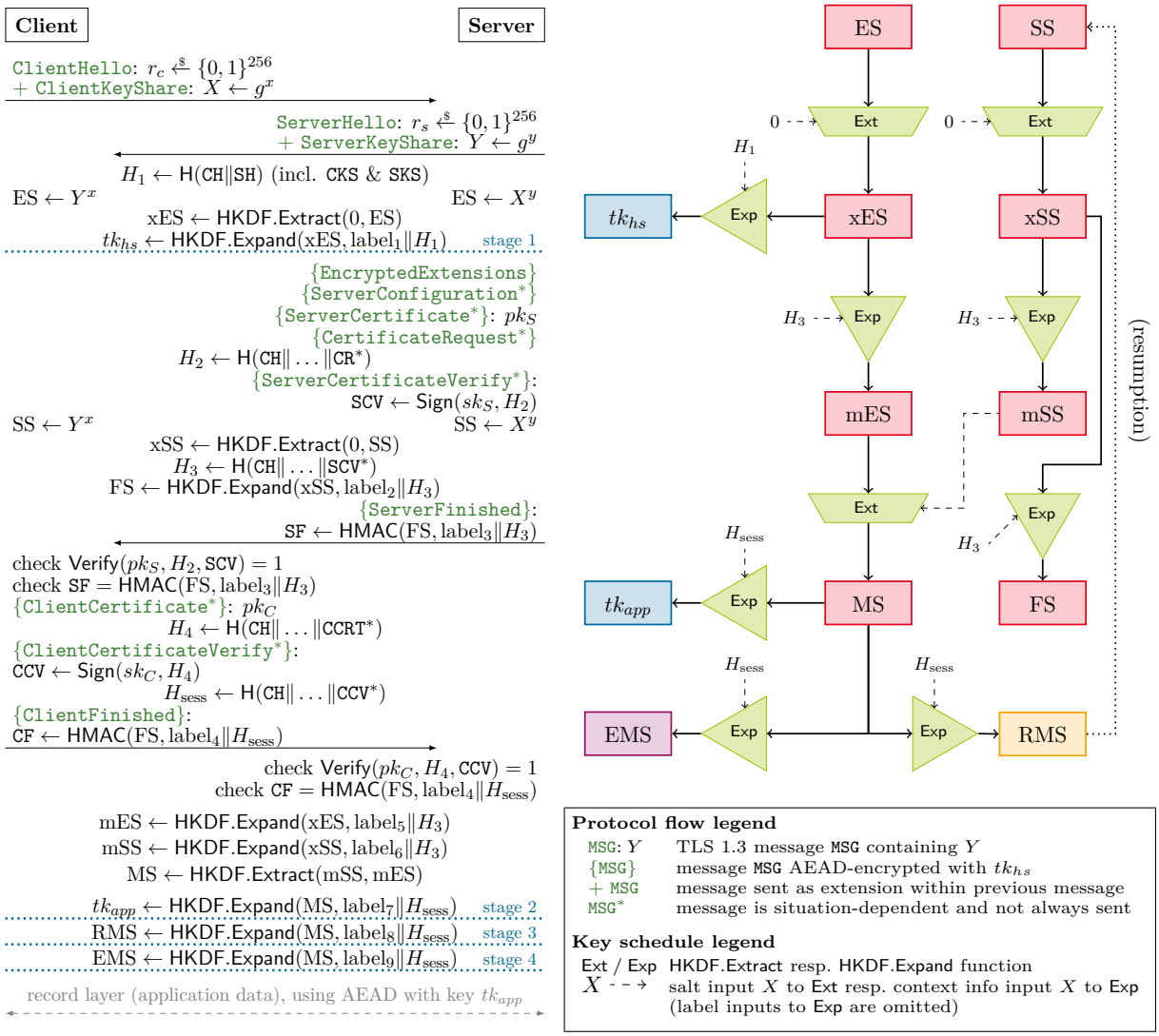


Figure 6.1: The TLS 1.3 `draft-10` full (EC)DHE handshake protocol (left) and key schedule (right).

include various extension fields, for the full (EC)DHE mode at least the following one must to be present.

- `ClientKeyShare` (`CKS`)/`ServerKeyShare` (`SKS`) are extensions sent within the `ClientHello` resp. `ServerHello` messages which contain the ephemeral Diffie–Hellman shares $X = g^x$ resp. $Y = g^y$ for one or more (in case of the client) groups.

Both parties can now compute the ephemeral secret ES as the Diffie–Hellman shared value g^{xy}. Key derivation is then done using HKDF in the extract-then-expand paradigm [Kra10, KE10], computing first an extracted value xES from which the handshake traffic key tk_{hs} is expanded; both are unauthenticated at this point. We adopt here the standard notation for the two HKDF functions as introduced in Section 3.2.3.

All subsequent messages are encrypted using tk_{hs}:

- `EncryptedExtensions` (`EE`), sent by the server, allows to specify further extensions.

- `ServerConfiguration` (`SC`) contains a server configuration (cryptographically a semi-static Diffie–Hellman share) which allows a client to later run an abbreviated (0-RTT) handshake (see the description of the Diffie–Hellman-based 0-RTT handshake in Section 7.4).
- `ServerCertificate` (`SCRT`)/`ClientCertificate` (`CCRT`) contain the public-key certificate of the respective party.
- `CertificateRequest` (`CR`) is sent by the server if it demands that the client authenticates using a certificate.
- `ServerCertificateVerify` (`SCV`)/`ClientCertificateVerify` (`CCV`) contain a digital signature over the *handshake hash* (the hash of all handshake messages sent and received at that point in the protocol run).
- `ClientFinished` (`CF`)/`ServerFinished` (`SF`) contain an HMAC message authentication code computed over the handshake hash keyed with the finished secret FS. The finished secret in turn is derived from the extracted version xSS of the static secret SS. While the static secret takes different values in other handshake variants, it equals the ephemeral secret (SS = ES) in the full (EC)DHE handshake.

Both parties can now compute the master secret MS (as an extraction of expanded ephemeral and static secrets). From the master secret, the application traffic key tk_{app} as well as the resumption master secret RMS for use in future session resumptions and the exporter master secret EMS allowing the potential derivation of further keying material are computed through HKDF expansion steps which include the final handshake hash value, which is also called the *session hash* H_{sess}.

On 0.5-RTT data and post-handshake messages. In our analysis of the TLS 1.3 handshake candidates we focus on the main components of the handshake and hence in particular do not capture the following two more advanced options specified or envisioned in the draft standards.

First, instead of deriving the application traffic key tk_{app} at the end of the handshake (as depicted in Figure 6.1), the server might already do so after sending the `ServerFinished` message in order to send so-called *0.5-RTT data* directly following his handshake messages, i.e., without waiting for the `ClientFinished` response. We omit analyzing this variant of the handshake but expect that results for it with potentially weaker authentication guarantees for tk_{app} can be obtained in our model.

Second, TLS 1.3 introduces *post-handshake messages* that can be sent encrypted under tk_{app} (potentially long) after the initial handshake was completed in order to update the used traffic key, authenticate the client, or issue tickets for session resumption.[21] Here, we focus on the main handshake and do not consider post-handshake messages.

6.3 Security of the TLS 1.3 draft-10 Full (EC)DHE Handshake

As discussed above, we employ our (public-key) multi-stage key exchange (MSKE) model (cf. Chapter 4) for the analysis of TLS 1.3's full (EC)DHE handshake mode in `draft-10`, which we shorten to `draft-10-(EC)DHE` here. We will establish that `draft-10-(EC)DHE` provides key independence and forward secrecy for all keys derived along with the following protocol-specific properties ($\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY}$):

[21] TLS 1.3 `draft-10` specifies only the last one, the `NewSessionTicket` message to issue resumption tickets. The other post-handshake messages were introduced in `draft-11` [Res15f].

- $\mathsf{M} = 4$: `draft-10-(EC)DHE` consists of four stages deriving the handshake and application traffic keys tk_{hs} and tk_{app}, the resumption master secret RMS, and the exporter master secret EMS (in this order).
- $\mathsf{AUTH} = \{(\mathsf{unauth}, \mathsf{auth}_{\mathrm{fin}}, \mathsf{auth}_{\mathrm{fin}}, \mathsf{auth}_{\mathrm{fin}}) \mid \mathsf{auth}_{\mathrm{fin}} \in \{\mathsf{unauth}, \mathsf{unilateral}, \mathsf{mutual}\}\}$: the first-stage key tk_{hs} is always unauthenticated, the remaining three keys (tk_{app}, RMS, and EMS) can be (all) either unauthenticated[22], unilaterally (server-only) authenticated, or mutually authenticated.
- $\mathsf{USE} = (\mathsf{internal}, \mathsf{external}, \mathsf{external}, \mathsf{external})$: The handshake traffic key is used internally within `draft-10-(EC)DHE` to encrypt the second part of the handshake; all other keys are not used within the main handshake.
- $\mathsf{REPLAY} = (\mathsf{nonreplayable}, \mathsf{nonreplayable}, \mathsf{nonreplayable}, \mathsf{nonreplayable})$: All stages' keys are non-replayable due to nonces included from both sides.

The `draft-10` full (EC)DHE handshake does not involve semi-static keys in its key derivation (but only possibly transmits them for a later 0-RTT handshake in a `ServerConfiguration` message). We hence do not have to treat such keys in the notation of our model and can thus ignore the NewSemiStaticKey and RevealSemiStaticKey queries in the following analysis.

Further, we need to define the session and contributive identifiers for the four stages of the TLS 1.3 `draft-10` full handshake. We let the session identifiers for the two stages deriving the handshake traffic key tk_{hs} and the application traffic key tk_{app} be the unencrypted messages sent and received excluding the finished messages:

$$\mathsf{sid}_1 = (\texttt{ClientHello}, \texttt{ClientKeyShare}, \texttt{ServerHello}, \texttt{ServerKeyShare}) \quad \text{and}$$
$$\mathsf{sid}_2 = (\texttt{ClientHello}, \texttt{ClientKeyShare}, \texttt{ServerHello}, \texttt{ServerKeyShare}, \texttt{EncryptedExtensions},$$
$$\texttt{ServerConfiguration}^*, \texttt{ServerCertificate}^*, \texttt{CertificateRequest}^*,$$
$$\texttt{ServerCertificateVerify}^*, \texttt{ClientCertificate}^*, \texttt{ClientCertificateVerify}^*).$$

Here, starred (*) components are not present in all authentication modes. We capture the further derived resumption master secret RMS and exporter master secret EMS in stages 3 and 4 and define the session identifier to be $\mathsf{sid}_3 = (\mathsf{sid}_2, \text{“RMS”})$ and $\mathsf{sid}_4 = (\mathsf{sid}_2, \text{“EMS”})$ which are uniquely determined by the second-stage identifier sid_2.

We stress that defining session identifiers over the *unencrypted* messages is necessary to obtain key-independent Multi-Stage security. Otherwise, we would need to either resort to key dependence, or guarantee that an adversary is not able to re-encrypt a sent message into a different ciphertext even if it knows the handshake traffic key tk_{hs} used (due to a Reveal query)—a property generally not to be expected from a (potentially randomized) encryption scheme.

Concerning the contributive identifiers, we let the client (resp. server) on sending (resp. receiving) the `ClientHello` and `ClientKeyShare` messages set $\mathsf{cid}_1 = \mathsf{sid}_1 = (\texttt{CH}, \texttt{CKS})$ and subsequently, on receiving (resp. sending) the `ServerHello` and `ServerKeyShare` messages, extend it to $\mathsf{cid}_1 = (\texttt{CH}, \texttt{CKS}, \texttt{SH}, \texttt{SKS})$. The other contributive identifiers are set to $\mathsf{cid}_i = \mathsf{sid}_1$ for stages $i \in \{2, 3, 4\}$ by each party on sending its respective `Finished` message.

We are now ready to state our formal security result for the TLS 1.3 `draft-10` full (EC)DHE handshake.

[22] The TLS 1.3 `draft-10` specification is not entirely clear whether a fully anonymous (EC)DHE handshake is permissible. We include this option in our analyses in this thesis for completeness, establishing that TLS 1.3 can achieve anonymous key exchange security.

Theorem 6.1 (Match security of `draft-10-(EC)DHE`)**.** *The TLS 1.3* `draft-10` *full (EC)DHE handshake is* Match*-secure with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. For any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq n_s^2 \cdot 1/q \cdot 2^{-|nonce|},$$

where n_s *is the maximum number of sessions,* q *is the group order, and* $|nonce| = 256$ *is the bit-length of the nonces.*

Match security follows from the way the session identifiers are chosen (to include all unencrypted messages), in particular guaranteeing that partnered sessions derive the same key, authenticity, and contributive identifiers. The given security bound takes into account the probability that two honest sessions choose the same nonce and group element.

Proof. We need to show the six properties of Match security (cf. Definition 4.1).

1. *Sessions with the same session identifier for some stage hold the same key at that stage.*
   For the first stage this follows as the session identifier contains the parties' Diffie–Hellman contributions g^x and g^y, which uniquely identify the Diffie–Hellman key, as well as all data entering the key derivation step. Hence, equal session identifiers imply that both parties compute the same ephemeral secret and the same session key on the first stage. For the second, third, and fourth stage note that the identifier sid_2 (and hence also sid_3 and sid_4) contains the full sid_1, implying that the parties have also computed the same ephemeral secret. Since the key derivation for the stages 2–4 is only based on this secret value (and the identical static secret SS = ES) and data from sid_2, it follows that the session keys must be equal, too.

2. *Sessions with the same session identifier for some stage agree on that stage's authentication level.*
   Observe that, for the first stage, the only admissible authenticity by design of TLS 1.3 is $\mathsf{auth}_1 = \mathsf{unauth}$ on which, hence, all sessions will agree. For the other stages, the exchanged messages (except for the finished messages) contained in the session identifier sid_2 (and hence also sid_3 and sid_4) uniquely determines the authenticity property for these stages. More precisely, according to the protocol specification, both sessions will agree on $\mathsf{sid}_2 =$ (`ClientHello`, `ClientKeyShare`, `ServerHello`, `ServerKeyShare`, `EncryptedExtensions`) if and only if both have $\mathsf{auth}_2 = \mathsf{unauth}$. If sid_2 additionally contains `ServerConfiguration`* (optional), `ServerCertificate`, and `ServerCertificateVerify`, they agree on $\mathsf{auth}_2 = \mathsf{unilateral}$. If it moreover contains messages `CertificateRequest`, `ClientCertificate`, and `ClientCertificateVerify`, the sessions agree on mutual authentication. Moreover, $\mathsf{auth}_2 = \mathsf{auth}_3 = \mathsf{auth}_4$ always holds hence same identifiers also imply agreement on authenticity.

3. *Sessions with the same session identifier for some stage share the same contributive identifier.*
   This holds since, for each stage, the contributive identifier value is final and equals sid_1 once the session identifier is set, and the messages in sid_1 are contained in every session identifier.

4. *Sessions are partnered with the intended (authenticated) participant.*
   First of all observe that this case only applies to unilaterally or mutually authenticated stages, hence the stages 2–4 only. In TLS 1.3, the client obtains the server's identity within the `ServerCertificate` message and vice versa the server obtains the client's identity

(in case of mutual authentication) within the `ClientCertificate` message. Moreover, honest clients and servers will not send a certificate attesting an identity different from their own. Hence, as both messages are contained in the session identifiers of stages 2–4 (in the respective authentication mode), agreement on sid_2 (and hence the same for sid_3, sid_4) implies agreement on the respective partner's identity.

5. *Session identifiers are distinct for different stages.*
   This holds trivially as sid_2 contains strictly more messages than sid_1 and sid_3 as well as sid_4 contain unique labels.

6. *At most two sessions have the same session identifier at any non-replayable stage.*
   Observe that the group element for the Diffie–Hellman key, as well as a random nonce, of both the initiator and the responder enter the session identifiers. Therefore, in order to have a threefold collision among session identifiers of honest parties, the third session would need to pick the same group element and nonce as one of the other two sessions. The probability that there exists such a collision can hence be bounded from above by $n_s^2 \cdot 1/q \cdot 2^{-|\text{nonce}|}$ where n_s is the maximum number of sessions, q is the group order, and $|\text{nonce}| = 256$ the nonces' bit-length. □

Theorem 6.2 (Multi-Stage security of `draft-10-(EC)DHE`)**.** *The TLS 1.3* `draft-10` *full (EC)DHE handshake is* Multi-Stage*-secure in a key-independent and stage-1-forward-secret manner with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. Formally, for any efficient adversary* $\mathcal{A}$ *against the* Multi-Stage *security there exist efficient algorithms* $\mathcal{B}_1, \ldots, \mathcal{B}_6$ *such that*

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq 4n_s \cdot \Big(\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2} + n_s \cdot \Big(\mathsf{Adv}^{\mathsf{snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_4} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_5} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6} \Big) \Big),$$

where n_s *is the maximum number of sessions and* n_u *is the maximum number of users.*

Multi-Stage security essentially follows from a combination of two lines of reasoning about the `draft-10` (EC)DHE handshake. First, (unforgeable) signatures covering the (collision-resistantly hashed) handshake messages ensure that, for authenticated stages, the exchanged Diffie–Hellman shares g^x and g^y originate from an honest partner session (client, resp. server). Then, from the joint Diffie–Hellman key g^{xy} unknown to the adversary, all keys are derived via HKDF and independently, allowing us to show that they are indistinguishable from random keys (under the snPRF-ODH assumption and according assumptions on the Extract and Expand steps of HKDF).

Proof. First of all we consider the case that the adversary $\mathcal{A}$ makes a single Test query only (and denote the corresponding game as 1-Multi-Stage). This reduces its advantage, based on a hybrid argument (which we delay for readability reasons to Lemma 6.3 below), by a factor at most $1/4n_s$ as there are four stages in each of the n_s sessions. We from now on can speak about *the* session label tested at stage i, which we know in advance.

In the following, we proceed via a sequence of games. Starting from the original Multi-Stage game (with a single Test query), we bound the advantage difference of adversary $\mathcal{A}$ between any two games by complexity-theoretic assumptions until we reach a game where the adversary $\mathcal{A}$ cannot win, i.e., its advantage is at most 0.

Game 0. The initial game equals the Multi-Stage game with a single Test query. Combined with the initial hybrid step we therefore have that

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq 4n_s \cdot \mathsf{Adv}^{G_0}_{\texttt{draft-10-(EC)DHE},\mathcal{A}}.$$

Game 1. In this game, we let the challenger abort the game if any two honest sessions compute the same hash value for different inputs in any evaluation of the hash function H.

Let $\mathsf{abort}_{\mathsf{H}}$ denote the event that the challenger aborts in this case. We can bound the probability $\Pr[\mathsf{abort}_{\mathsf{H}}]$ by the advantage $\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}$ of an adversary $\mathcal{B}_1$ against the collision resistance of the hash function H. To this extent, $\mathcal{B}_1$ acts as the challenger in Game 0, using its description of H to compute hash values, and running adversary $\mathcal{A}$ as a subroutine. If the event $\mathsf{abort}_{\mathsf{H}}$ occurs, $\mathcal{B}_1$ outputs the two distinct input values to H resulting in the same hash value as a collision.

Note that $\mathcal{B}_1$ perfectly emulates the attack of $\mathcal{A}$ according to G_0 up to the point where a hash collision occurs. As $\mathcal{B}_1$ wins if $\mathsf{abort}_{\mathsf{H}}$ is triggered, we have that $\Pr[\mathsf{abort}_{\mathsf{H}}] \leq \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}$ and thus

$$\mathsf{Adv}^{G_0}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_1}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

Our security analysis now separately considers the two (disjoint) cases that

A. the adversary tests a (client or server)[23] session without honest contributive partner in the first stage (i.e., for the test session label there exists no $\mathsf{label}' \neq \mathsf{label}$ with $\mathsf{label}.\mathsf{cid}_1 = \mathsf{label}'.\mathsf{cid}_1$), and

B. the tested session has an honest contributive partner in stage 1 (i.e., there exists label' with $\mathsf{label}.\mathsf{cid}_1 = \mathsf{label}'.\mathsf{cid}_1$).

This allows us to split the adversary's advantage along these two cases:

$$\mathsf{Adv}^{G_1}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_1,\text{test without partner}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{G_1,\text{test with partner}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}}.$$

Case A. Test without Partner

We first consider the case that the tested session label (which might be a client/initiator or a server/responder session) is without honest contributive partner in the first stage. Since cid_1 is contained in sid_1, we know that label also has no session partner in stage 1 (i.e., there is no other label' with $\mathsf{label}.\mathsf{sid}_1 = \mathsf{label}'.\mathsf{sid}_1$). Having an honest partner in the second (or later) stage implies having also one in the first stage (as $\mathsf{sid}_1 = \mathsf{cid}_2 = \mathsf{cid}_3 = \mathsf{cid}_4$), hence the tested session must actually be without honest partner in all stages. Observe that, by definition of the model, the adversary cannot win in this case if the tested key is not authenticated by the peer session. Hence, we can assume that, if the test session is a client session then the key is responder-authenticated (i.e., $\mathsf{label}.\mathsf{auth}_i \in \{\mathsf{unilateral}, \mathsf{mutual}\}$), respectively if the test session is a server session then the key is initiator-authenticated (i.e., $\mathsf{label}.\mathsf{auth}_i = \mathsf{mutual}$). This allows us to focus on Test queries in the stages 2–4 according to the authentication properties AUTH provided.

Game A.0. This initial game equals Game 1 above where the adversary is, by our assumption, restricted to test a session without honest contributive partner in the first stage. Therefore,

$$\mathsf{Adv}^{G_{A.0}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} = \mathsf{Adv}^{G_1,\text{test without partner}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}}.$$

Game A.1. In this game, we let the challenger abort if the tested session receives, within the `ServerCertificateVerify` or `ClientCertificateVerify` message, a valid signature under the public key pk_U of some user $U \in \mathcal{U}$ such that the hash value message has *not* been signed

[23] The original analysis in [DFGS16] treated the client and server sub-cases separately; merging them here allows us to slightly optimize the overall security bound.

by any of the honest sessions. Recall that due to the peer authentication of the tested session's stage i such `CertificateVerify` message must be received.

Let $\mathsf{abort}_{\mathsf{Sig}}$ denote the event that the challenger aborts in this case. We bound the probability $\Pr[\mathsf{abort}_{\mathsf{Sig}}]$ of its occurrence by the advantage of an adversary $\mathcal{B}_2$ against the EUF-CMA security of the signature scheme Sig, denoted $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2}$. In the reduction, $\mathcal{B}_2$ first guesses a user $U \in \mathcal{U}$ which it associates with the challenge public key pk^* in the EUF-CMA game, then generates all long-term key pairs for the other users $U' \in \mathcal{U} \setminus \{U\}$ and runs the Multi-Stage game $G_{A.0}$ for $\mathcal{A}$, including potentially an abort due to hash collisions (cf. Game 1). For any signature to generate for user U in honest sessions for a hash value, $\mathcal{B}_2$ calls its signing oracle about the hash value. When $\mathsf{abort}_{\mathsf{Sig}}$ is triggered, $\mathcal{B}_2$ outputs the signature the tested session received together with the hash value as a forgery.[24]

Since every honest session has a different session identifier than the tested session in the first stage (as the latter has no partnered session in this stage), no honest party will seek to sign the transcript value expected by the tested session. Moreover, by the modification in Game 1, there is no collision between any two honest evaluations of the hash function, so in particular there is none for the hash value computed by the tested session, implying that the hash value in question has not been signed by an honest party before. If $\mathcal{B}_2$ correctly guessed the user under whose public key the obtained signature verifies, that signature output by $\mathcal{B}_2$ is a valid forgery in the sense that its message was never queried to the EUF-CMA oracle before. Hence, $\mathcal{B}_2$ wins if $\mathsf{abort}_{\mathsf{Sig}}$ occurs and it has guessed the correct user amongst the set of (at most) n_u users and we have that $\Pr[\mathsf{abort}_{\mathsf{Sig}}] \leq n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2}$ and thus

$$\mathsf{Adv}^{G_{A.0}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{A.1}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2}.$$

Now, if Game A.1 does not abort, we are assured that an honest session issued a signature on the (hashed) messages the test session expects within the `CertificateVerify` message. This signature is computed over $H_2 = \mathsf{H}(\texttt{CH}, \texttt{CKS}, \texttt{SH}, \texttt{SKS}, \texttt{EE}, \texttt{SC}^*, \texttt{SCRT}, \texttt{CR}^*)$ in case of `ServerCertificateVerify` resp. $H_4 = \mathsf{H}(\texttt{CH}, \texttt{CKS}, \texttt{SH}, \texttt{SKS}, \texttt{EE}, \texttt{SC}^*, \texttt{SCRT}, \texttt{CR}^*, \texttt{SF}, \texttt{CCRT})$ in case of `ClientCertificateVerify`, i.e., in particular contains all messages in sid_1. Hence, the tested session and the honest session outputting the signature agree on sid_1, so also on cid_1, and are hence (contributively) partnered in the first stage.

The adversary $\mathcal{A}$ therefore cannot test a session without honest first-stage partner in Game A.1 anymore, resulting in the test bit b_{test} being unknown to $\mathcal{A}$ and hence

$$\mathsf{Adv}^{G_{A.1}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq 0.$$

Case B. Test with Partner

In the second case, the tested session (client or server) has an honest contributive partner in the first stage, i.e., we know there exists another label' such that $\mathsf{label}.\mathsf{cid}_1 = \mathsf{label}'.\mathsf{cid}_1$. This allows Test queries to be potentially issued in any of the four stages.

Game B.0. We start with an initial game equal to Game 1 above, but restrict the adversary to only test a session having an honest contributive partner in the first stage in order to have

$$\mathsf{Adv}^{G_{B.0}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} = \mathsf{Adv}^{G_1,\text{test with partner}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}}.$$

Game B.1. First, we guess a session $\mathsf{label}' \neq \mathsf{label}$ (among the at most n_s sessions in the game) and abort the game in case this session is not an honest contributive partner (in stage 1)

[24] Note that, although the `CertificateVerify` message containing the signature is sent encrypted, the honest tested session is simulated by $\mathcal{B}_2$ and hence $\mathcal{B}_2$ can in particular decrypt this message.

of the tested session, i.e., we abort if $\mathsf{label.cid}_1 \neq \mathsf{label}'.\mathsf{cid}_1$. Note that we can assume that $\mathcal{A}$ always issues a Test query, as this cannot decrease the adversary's advantage. The guessing strategy then reduces the adversary's advantage by a factor of at most $1/n_s$.

$$\mathsf{Adv}^{G_{B.0}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq n_s \cdot \mathsf{Adv}^{G_{B.1}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}}.$$

From now on, we can speak of *the* session label' (contributively) partnered with the tested session label in stage 1 and know label' in advance.

Game B.2. At this point, having the (honest) contributions to the tested session fixed, we can encode a Diffie–Hellman challenge in the shares g^x and g^y at the tested session. If a client session is tested, we know that the partnered session label' guessed in Game B.1 holds the same shares. However, if a server session is tested, the client session label' may obtain a modified (and potentially adversarially-known) value $g^{y'}$ in the `ServerHello` message. In order to be able to compute the ephemeral secret ES of session label' (and correctly answer to a Reveal query on derived keys) without knowing exponents x or y', we employ the PRF-ODH assumption in its snPRF-ODH variant [JKSS12, BFGJ17] here (see Definition 3.6 in Section 3.2.2 for its definition). More specifically, we assume HKDF.Extract satisfies the snPRF-ODH assumption when considered as PRF deriving xES and xSS using ES = SS as key and salt 0 as label.

In Game B.2, we then replace the extracted ephemeral and static secrets xES and xSS (which are equal as ES = SS) by a uniform and independent random string $\widetilde{\mathrm{xES}} = \widetilde{\mathrm{xSS}} \xleftarrow{\$} \{0,1\}^\lambda$ in the tested session and, if derived there, in the partnered session. We bound the introduced advantage difference for $\mathcal{A}$ by the advantage of an algorithm $\mathcal{B}_3$ against the snPRF-ODH security of HKDF.Extract (using ES = SS as source key material and 0 as salt) as follows. First, $\mathcal{B}_3$ outputs 0 as the PRF challenge label. It obtains Diffie–Hellman shares g^x and g^y which it encodes in the `ClientKeyShare` and `ServerKeyShare` message, respectively, of the tested and contributively partnered session label and label'. It further obtains a PRF challenge value which it uses as the extracted ephemeral and static secret xES = xSS in session label and, if using the same Diffie–Hellman shares, session label'. In case label' is a client session and obtains within `ServerKeyShare` a value $g^{y'} \neq g^y$, $\mathcal{B}_3$ uses its single ODH_u query in the snPRF-ODH game to compute $\mathrm{xES} = \mathrm{xSS} \leftarrow \mathsf{HKDF.Extract}(0, g^{xy'})$.

The simulation $\mathcal{B}_3$ provides equals Game B.1 in case the PRF challenge value equals $\mathsf{HKDF.Extract}(0, g^{xy})$ and Game $B.2$ if the challenge is a uniformly random value. Thus,

$$\mathsf{Adv}^{G_{B.1}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.2}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_3}.$$

Game B.3. Next, we replace the handshake traffic key tk_{hs}, the expanded ephemeral and static secrets mES and mSS, and the finished secret FS derived in both the tested and its partnered session by independent uniformly random values $\widetilde{tk_{hs}}, \widetilde{\mathrm{mES}}, \widetilde{\mathrm{mSS}}, \widetilde{\mathrm{FS}} \xleftarrow{\$} \{0,1\}^\lambda$. Recall that in contrast to the extracted secrets xES and xSS that are derived using the same salt, the expanded secrets mES and mSS are computed using distinct labels.

We can bound the difference in $\mathcal{A}$'s advantage introduced through this step by the security of the HKDF.Expand function which we model as a pseudorandom function keyed with uniformly random bit strings from $\{0,1\}^\lambda$. The reduction $\mathcal{B}_4$ uses its PRF oracle for the evaluations of HKDF.Expand under the key $\widetilde{\mathrm{xES}} = \widetilde{\mathrm{xSS}}$ in the tested and its partnered session. Observe that, in case the oracle computes the PRF function, this equals Game B.2, whereas, if it computes a random function, this equals Game B.3. The simulation is sound because the extracted ephemeral and static secret $\widetilde{\mathrm{xES}} = \widetilde{\mathrm{xSS}}$, by the change in Game B.2, is a random bit string chosen independently of all other values in the game.

The advantage of $\mathcal{B}_4$ in the PRF security game therefore bounds the advantage difference such that

$$\mathsf{Adv}^{G_{B.2}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.3}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_4}.$$

Game B.4. We now replace the master secret MS by a uniformly random value $\widetilde{\mathrm{MS}}$. This again can be bounded by the advantage against the PRF security (with uniformly random keys) of HKDF.Extract as MS is derived from key $\widetilde{\mathrm{mES}}$ and salt $\widetilde{\mathrm{mSS}}$, now independent uniformly random bit strings. Therefore,

$$\mathsf{Adv}^{G_{B.3}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.4}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_5}.$$

Game B.5. Finally, we replace all HKDF.Expand evaluations using the (replaced) master secret $\widetilde{\mathrm{MS}}$ as key in the tested and its partnered session by a (lazy-sampled) random function. This change affects the derivation of the handshake traffic key tk_{app}, the resumption master secret RMS, and the exporter master secret EMS which are hereby replaced with independent random values $\widetilde{tk_{app}}, \widetilde{\mathrm{RMS}}, \widetilde{\mathrm{EMS}} \xleftarrow{\$} \{0,1\}^\lambda$ in the tested session and, for same computations, also its partnered session.

As in the previous steps, we can bound the difference in $\mathcal{A}$'s advantage introduced through this step by the PRF security of HKDF.Expand, again defined for keys being uniformly random bit strings from $\{0,1\}^\lambda$. To this extent, the reduction $\mathcal{B}_6$ as above uses its PRF oracle for all evaluations of HKDF.Expand under the key $\widetilde{\mathrm{MS}}$ in the tested and its partnered session. Depending on the oracles behavior, this perfectly simulates either Game B.4 or Game B.5, as $\widetilde{\mathrm{MS}}$ is a uniformly random and independent bit string and different labels are used in the derivation of tk_{app}, RMS, and EMS.

We can hence can infer that

$$\mathsf{Adv}^{G_{B.4}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.5}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6}.$$

In Game B.5, the session keys $\widetilde{tk_{hs}}$ and $\widetilde{tk_{app}}$ as well as the resumption and exporter master secrets $\widetilde{\mathrm{RMS}}$ and $\widetilde{\mathrm{EMS}}$ are now chosen independently and uniformly at random. Note furthermore that the adversary cannot reveal the partnered session without immediately losing the game and cannot render the first-stage contributively partnered to derive the same keys while not being partnered, as the (hashed) session identifier messages fully enter the key derivation in each stage. As the response to its Test query is hence independent of the test bit b_{test}, the adversary $\mathcal{A}$ cannot distinguish whether it is given the real key or (another) independently chosen random value and thus

$$\mathsf{Adv}^{G_{B.5}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq 0.$$

Combining the various bounds implied by the above sequence of game transitions yields the stated security bound. □

We complete our analysis of the TLS 1.3 `draft-10` full (EC)DHE handshake by providing the details of the hybrid argument showing that if we restrict the adversary in Theorem 6.2 to a single Test query, this reduces its advantage by a factor at most $1/4n_s$ (for the four stages in each of the n_s sessions).

Lemma 6.3. *Restricting the adversary $\mathcal{A}$ in Theorem 6.2 to a single* Test *query reduces its advantage by a factor at most $1/4n_s$ (for the four stages in each of the n_s sessions). Formally, there exists an efficient algorithm $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq 4n_s \cdot \mathsf{Adv}^{\text{1-Multi-Stage}}_{\texttt{draft-10-(EC)DHE},\mathcal{B}}.$$

Proof. The hybrid argument consists of a sequence of games G_n for $n = 0, \ldots, 4n_s$, where G_n behaves like $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}}$ except that the first n tested keys are the actual derived keys, and the remaining ones are replaced by random ones (uniformly chosen from $\mathcal{D}$). Here, however, we assume consistent replacements in the sense that a Test query returns the previously returned key if a partnered session has already been tested. In particular, if for a tested session there is a partner session among the first n tested sessions, then we return the actual derived key, even if the now tested session comes after the n's Test query. Note that, by construction, identical session identifiers yield identical keys, such that we cannot generate inconsistencies by having partners (with identical identifiers) but different keys. Also observe that G_{4n_s} equals the unmodified game $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}}$ with test bit $b_{\mathsf{test}} = 0$ even if the adversary makes less than $4n_s$ Test queries, and that in G_0 all keys are chosen uniformly at random (but consistent over partnered sessions). This means that G_0 is identical to $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}}$ with test bit $b_{\mathsf{test}} = 1$.

For the hybrid argument we construct a reduction $\mathcal{B}$ to the single-test $\mathsf{Multi\text{-}Stage}$ game $G^{\text{1-}\mathsf{Multi\text{-}Stage}}_{\texttt{draft-10-(EC)DHE},\mathcal{B}}$ as follows. Initially $\mathcal{B}$ chooses an index n at random between 1 and $4n_s$. It initializes a counter c to $c = 0$ (indicating the number of tested session keys replaced by random values so far) as well as (initially empty) sets $\mathsf{SKEY}_1, \mathsf{SKEY}_2, \mathsf{SKEY}_2^e, \mathsf{SKEY}_3, \mathsf{SKEY}_3^e$, $\mathsf{SKEY}_4, \mathsf{SKEY}_4^e \subseteq \{0,1\}^* \times [\mathcal{D}]$ for identifiers and keys in the support $[\mathcal{D}]$ of $\mathcal{D}$, to keep track of established session keys for a consistent simulation.[25] Basically, SKEY_1 corresponds to session identifier–key pairs of the first stage which $\mathcal{B}$ has already collected, similarly SKEY_2, SKEY_3, and SKEY_4 are for the second, third, and fourth stage, and SKEY_2^e, SKEY_3^e, resp. SKEY_4^e for transcripts and second-stage, third-stage, resp. fourth-stage keys where $\mathcal{B}$ cannot decrypt the data in the session identifier (yet), since the data entering the identifier are encrypted for transmission. We usually write sid_1 and $\mathsf{sid}_2 = (\mathsf{sid}_1, \mathsf{sid}_{+2})$ for the session identifiers for the first and second stage, respectively, denoting the second part of the stage-two identifier as sid_{+2}; we also write $\{\mathsf{sid}_{+2}\}$ to denote the fact that $\mathcal{B}$ only holds an encrypted version of the second part. Recall that $\mathsf{sid}_3 = (\mathsf{sid}_2, \text{“RMS”})$ and $\mathsf{sid}_4 = (\mathsf{sid}_2, \text{“EMS”})$, hence we can also write them as $\mathsf{sid}_3 = (\mathsf{sid}_1, \mathsf{sid}_{+2}, \text{“RMS”})$ resp. $\mathsf{sid}_4 = (\mathsf{sid}_1, \mathsf{sid}_{+2}, \text{“EMS”})$. All sets SKEY_1, SKEY_2, SKEY_3, and SKEY_4 will be (individually and together) consistent during the entire simulation in the sense that they do not contain entries with identical session identifiers but different keys. This is clearly true upon initialization and remains so whenever we add elements to either set.

Algorithm $\mathcal{B}$ then runs $\mathcal{A}$, relaying all queries and answers of $\mathcal{A}$ to its external oracles, with one exception: if $\mathcal{A}$ makes one of its multiple Test queries (where we can assume that all such queries are made for accepted executions only), then $\mathcal{B}$ increments c and checks c against n.

- If $c < n$ then $\mathcal{B}$ simply makes a Reveal query for the corresponding stage and returns the obtained key key. Additionally, $\mathcal{B}$ does the following updates to its lists SKEY_1, SKEY_2, SKEY_2^e, SKEY_3, SKEY_3^e, SKEY_4, and SKEY_4^e.

  If the inspected session is a stage-one session with identifier sid_1 then $\mathcal{B}$ places the session identifier and the returned key value into SKEY_1. Then, for each element $(\mathsf{sid}'_1, \{\mathsf{sid}'_{+2}\})$ in SKEY_2^e which carries $\mathsf{sid}'_1 = \mathsf{sid}_1$ as part of the session identifier, use the session key for sid_1 to recover the (decrypted) identifier sid'_2 for the entry and put the identifier sid'_2 with its key into SKEY_2. Proceed analogously for each element $(\mathsf{sid}_{1'}, \{\mathsf{sid}'_{+2}\}, \text{“RMS”})$ in SKEY_3^e and $(\mathsf{sid}_{1'}, \{\mathsf{sid}'_{+2}\}, \text{“EMS”})$ in SKEY_4^e with $\mathsf{sid}'_1 = \mathsf{sid}_1$.

  If the inspected session is a stage-two session then $\mathcal{B}$ can recover the stage-two identifier $(\mathsf{sid}_1, \{\mathsf{sid}_{+2}\})$ (with some stage-one identifier sid_1 and some encrypted part). It checks

[25] As we will see, we can, by construction, always decide partnering in stages 3 and 4 (deriving the resumption master secret RMS resp. exporter master secret EMS) whenever we can decide it in stage 2. We will nevertheless state the according computations explicitly here for completeness.

if the stage-one identifier part sid_1 (in clear) is already in SKEY_1.[26] If so, then use the session key of the sid_1 entry to recover the full identifier $\mathsf{sid}_2 = (\mathsf{sid}_1, \mathsf{sid}_{+2})$ of the inspected session in clear, and put sid_2 with the key into SKEY_2. If there is no sid_1 entry in SKEY_1 then put the partly encrypted session identifier $(\mathsf{sid}_1, \{\mathsf{sid}_{+2}\})$ with the returned key into SKEY_2^e.

For third-stage and fourth-stage queries, proceed as for stage 2 with the according identifier $(\mathsf{sid}_1, \{\mathsf{sid}_{+2}\}, \text{“RMS”})$ resp. $(\mathsf{sid}_1, \{\mathsf{sid}_{+2}\}, \text{“EMS”})$.

Note that all four cases cannot introduce any inconsistencies in SKEY_1, SKEY_2, SKEY_3, or SKEY_4 as the derived and revealed key is uniquely determined given sid_1, sid_2, sid_3, resp. sid_4.

- If $c = n$ then $\mathcal{B}$ proceeds as follows. Algorithm $\mathcal{B}$ first extracts the session identifier of the tested session. This is trivial for a first-stage identifier sid_1 as it consists of the communication data in clear. For a second-, third-, or fourth-stage identifier $\mathsf{sid}_2 = (\mathsf{sid}_1, \{\mathsf{sid}_{+2}\})$, $\mathsf{sid}_3 = (\mathsf{sid}_1, \{\mathsf{sid}_{+2}\}, \text{“RMS”})$, resp. $\mathsf{sid}_4 = (\mathsf{sid}_1, \{\mathsf{sid}_{+2}\}, \text{“EMS”})$, which also contains messages sent encrypted under the first-stage handshake traffic key, algorithm $\mathcal{B}$ will make a Reveal query for the first stage of the test session to get the handshake traffic key. It puts the corresponding session identifier sid_1 and the revealed key into SKEY_1 for future reference. Revealing this first-stage session key is admissible due to key independence, despite the tested key in stage two, three, or four; it cannot force $\mathcal{B}$ with its single Test query to lose. At the same time it allows $\mathcal{B}$ to decrypt and recover the values for sid_{+2} in clear. Algorithm $\mathcal{B}$ also checks if one can now decrypt and move any entries in SKEY_2^e to SKEY_2, from SKEY_3^e to SKEY_3, resp. from SKEY_4^e to SKEY_4 (by checking for entries in SKEY_2^e, SKEY_3^e, resp. SKEY_4^e which carry the same stage-one identifier sid_1).

  Given that $\mathcal{B}$ now holds the session identifier (in clear) it can check if there already exists an entry in SKEY_1, SKEY_2, SKEY_3, or SKEY_4. If so, it returns the corresponding stored key. Else, $\mathcal{B}$ uses its single external Test query to get a key key, adds this key with the recovered identifier to the corresponding set SKEY_1, SKEY_2, SKEY_3, or SKEY_4, and returns the key to $\mathcal{A}$. Note that here SKEY_1, SKEY_2, SKEY_3, and SKEY_4 are still consistent in any case as $\mathcal{B}$, if at all, adds a new session identifier.

- If $c > n$ then $\mathcal{B}$ first recovers the session identifier of the requested test session. For a stage-one identifier sid_1 this is again easy by inspecting the communication so far. For a stage-two, stage-three, or stage-four identifier $\mathsf{sid}_2 = (\mathsf{sid}_1, \{\mathsf{sid}_{+2}\})$, $\mathsf{sid}_3 = (\mathsf{sid}_1, \{\mathsf{sid}_{+2}\}, \text{“RMS”})$, resp. $\mathsf{sid}_4 = (\mathsf{sid}_1, \{\mathsf{sid}_{+2}\}, \text{“EMS”})$ algorithm $\mathcal{B}$ first checks if there already exists an entry in SKEY_1 for the contained stage-one part sid_1 of the identifier of the inspected session and, if so, uses it to recover the full (unencrypted) sid_{+2} part of the identifier. If there is no entry then $\mathcal{B}$ makes a Reveal query for the stage-one key to again recover the full identifier sid_2, sid_3, resp. sid_4 and places sid_1 and the returned key into SKEY_1. Note that such a Reveal query cannot infringe with $\mathcal{B}$'s single Test query, because either the Test query was for a stage-two, -three, or -four session (and key independence enables us to reveal any stage-one key then), or the Test query was for a stage-one identifier in which case it must already be included in SKEY_1 and the Reveal query is not made.

  Given that $\mathcal{B}$ now knows the session identifier of the requested test session it checks if there is already an entry in SKEY_1, SKEY_2, SKEY_3, or SKEY_4 for it. If so, it returns the same key as in the entry. Else it picks a key key at random from $\mathcal{D}$, returns it to $\mathcal{A}$, and adds the obtained session identifier with the key value key to the corresponding set SKEY_1,

[26] Recall that any such entry would be unique.

SKEY_2, SKEY_3, or SKEY_4. Note again that there cannot exist such an entry in the lists if $\mathcal{B}$ adds some value, such that consistency remains intact.

Note that $\mathcal{B}$ provides a consistent simulation as any pair of Test queries for partnered sessions return identical answers: For Test queries of $\mathcal{A}$ for partnered sessions, both with $c < n$, this is clear as the Reveal queries make $\mathcal{B}$ return consistent keys. If the second query is for $c = n$ then $\mathcal{B}$ either has the identifier already in SKEY_1, SKEY_2, SKEY_3, or SKEY_4 and answers consistently, or the values for the first Test query are at least in SKEY_2^e, SKEY_3^e, or SKEY_4^e and are now moved to SKEY_2, SKEY_3, resp. SKEY_4 because $\mathcal{B}$ learns the key to sid_1 and first checks membership in SKEY_1, SKEY_2, SKEY_3, or SKEY_4 before possibly making the Test query. If the second Test query is for $c > n$ then the same argument as in the previous case applies. The latter is also true if the first Test query has been for $c = n$ or for $c > n$, because then the session identifier will be in SKEY_1, SKEY_2, SKEY_3, or SKEY_4 already.

Up to the finalization step $\mathcal{B}$'s simulation is perfect (except for potentially the state of the lost flag, see below). In particular, $\mathcal{B}$ loses according to the lost flag, either set during the processing of a Test query or in the finalization step, only if $\mathcal{A}$ in the simulation (and thus in a genuine execution) would lose. Conversely, as is, it can happen that $\mathcal{B}$ even avoids a loss which $\mathcal{A}$ would trigger with a Test query for a revealed partner, but $\mathcal{B}$ omits this Test query since it provides the answer differently. This corresponds to the finalize condition of Definition 4.2. Remarkably, this causes the following problem: if $\mathcal{A}$ decides to create a difference between the two cases, genuine keys or random ones in Test queries, by deliberately losing via, say, a Reveal query for a tested partner, this difference could vanish in $\mathcal{B}$'s simulation. In order to avoid this, we let $\mathcal{B}$ eventually run the internal finalization step and check if $\mathcal{A}$ loses (and if so, forcing a loss in its simulation by making a Reveal query to the same key the Test query was issued on).

To check for the condition in the finalization step note that all Test requests of $\mathcal{A}$ insert some values in the sets SKEY_1, SKEY_2, SKEY_2^e, SKEY_3, SKEY_3^e, SKEY_4, or SKEY_4^e. Only for those entries in SKEY_2^e, SKEY_3^e, and SKEY_4^e algorithm $\mathcal{B}$ cannot (yet) recover the session identifier; in particular there is no entry in SKEY_1 for those values, else they would have been moved to SKEY_2, SKEY_3, SKEY_4 already. Algorithm $\mathcal{B}$ can now "clean up" the sets SKEY_2^e, SKEY_3^e, and SKEY_4^e and move all entries to SKEY_2, SKEY_3, resp. SKEY_4, by making a-posteriori Reveal queries for the first-stage keys for all sessions in SKEY_2^e, SKEY_3^e, and SKEY_4^e to get the session key which allows us to decrypt the stage-two, stage-three, and stage-four identifier. These Reveal queries cannot force $\mathcal{B}$ to lose as the session identifier of the single tested session must be different (otherwise there would be an entry in SKEY_1). So we can from now on assume that $\mathcal{B}$ knows all session identifiers of $\mathcal{A}$'s requested test sessions in clear, and holds candidates for all first-stage keys of the tested sessions.

It remains that $\mathcal{B}$ checks the condition of the finalization (i.e., that $\mathcal{A}$ has not made a Reveal query to a partner of a tested session) as follows. Algorithm $\mathcal{B}$ recovers all the session identifiers of the revealed sessions (excluding the Reveal queries which only $\mathcal{B}$ made). For a stage-one Reveal request this is trivial, for a stage-two, -three, or -four request (with partial identifier sid_1) algorithm $\mathcal{B}$ checks if sid_1 appears among the tested sessions. If not, then this Reveal query clearly does not infringe with the Test queries. If it does appear, however, then we already have the first-stage key for sid_1 and can recover the full session identifier of the Reveal query and compare it to the set of tested sessions. If and only if $\mathcal{B}$ finds some match for some Reveal query then it forces a loss in its game.

Next, we check the losing condition within the Test query triggered when $\mathcal{A}$ requests some test such that another honest execution has already used this session key (in which case the adversary could potentially distinguish a random key). This check is easy to perform for $\mathcal{B}$ because, in handling the Test query, it always establishes the according session identifier sid_i of

the tested session's stage i. Hence, $\mathcal{B}$ can simply check whether there exists a partnered session in stage i whose execution state is already beyond $\mathsf{accepted}_i$ and force a loss in this case.

To check the condition within the Test query that $\mathcal{A}$ has not tested a session for which the partner is unauthenticated but which does not have an honest contributive partner, $\mathcal{B}$ can, for the first stage, simply inspect the the transcript as the elements of the contributive identifier cid_1 for the first stage are sent in clear. For stages 2–4 recall that $\mathsf{cid}_2 = \mathsf{cid}_3 = \mathsf{cid}_4 = \mathsf{sid}_1$ and hence $\mathcal{B}$ can again leverage the established session identifier of the tested session's stage to check if there exists an honest contributive partner for these stages upon finalization. In each case, if no contributive partnered session exists, then $\mathcal{B}$ provokes a loss.

With the final checks we have made sure that $\mathcal{B}$ loses due to some inadmissible query if and only if $\mathcal{A}$ would in the real attack. In particular, it follows that for fixed $n = 0$ the simulation of $\mathcal{B}$ has exactly the same success probability as $\mathcal{A}$ in game G_0, and analogously for $n = 4n_s$. A standard counting argument, basically considering the conditional probabilities for fixed choices of n, now shows that the advantage of $\mathcal{A}$ is at most a factor $4n_s$ of the advantage of $\mathcal{B}$. More formally, noting that for some fixed n and test bit $b_{\mathsf{test}} = 0$ we actually run the game for $b_{\mathsf{test}} = 1$ and $n - 1$, we obtain:

$$
\begin{aligned}
&\Pr\left[G^{\text{1-Multi-Stage}}_{\texttt{draft-10-(EC)DHE},\mathcal{B}} = 1 \middle| b_{\mathsf{test}} = 1\right] - \Pr\left[G^{\text{1-Multi-Stage}}_{\texttt{draft-10-(EC)DHE},\mathcal{B}} = 1 \middle| b_{\mathsf{test}} = 0\right] \\
&= \frac{1}{4n_s} \cdot \sum_{n_0=1}^{4n_s} \left(\Pr\left[G^{\text{1-Multi-Stage}}_{\texttt{draft-10-(EC)DHE},\mathcal{B}} = 1 \middle| n = n_0, b_{\mathsf{test}} = 1\right] - \Pr\left[G^{\text{1-Multi-Stage}}_{\texttt{draft-10-(EC)DHE},\mathcal{B}} = 1 \middle| n = n_0, b_{\mathsf{test}} = 0\right]\right) \\
&= \frac{1}{4n_s} \cdot \sum_{n_0=1}^{4n_s} \left(\Pr\left[G^{\text{1-Multi-Stage}}_{\texttt{draft-10-(EC)DHE},\mathcal{B}} = 1 \middle| n = n_0, b_{\mathsf{test}} = 1\right] - \Pr\left[G^{\text{1-Multi-Stage}}_{\texttt{draft-10-(EC)DHE},\mathcal{B}} = 1 \middle| n = n_0 - 1, b_{\mathsf{test}} = 1\right]\right) \\
&= \frac{1}{4n_s} \cdot \left(\Pr\left[G^{\text{1-Multi-Stage}}_{\texttt{draft-10-(EC)DHE},\mathcal{B}} = 1 \middle| n = 4n_s, b_{\mathsf{test}} = 1\right] - \Pr\left[G^{\text{1-Multi-Stage}}_{\texttt{draft-10-(EC)DHE},\mathcal{B}} = 1 \middle| n = 0, b_{\mathsf{test}} = 1\right]\right) \\
&= \frac{1}{4n_s} \cdot \left(\Pr\left[G_{4n_s} = 1\right] - \Pr\left[G_0 = 1\right]\right).
\end{aligned}
$$

Noticing that the first and last differences of probabilities in both cases, for $\mathcal{B}$ and for $\mathcal{A}$, correspond to $2 \cdot \mathsf{Adv}^{\text{1-Multi-Stage}}_{\texttt{draft-10-(EC)DHE},\mathcal{B}}$ and $2 \cdot \mathsf{Adv}^{\text{Multi-Stage},\mathcal{D}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}}$, the claim follows. □

6.4 The TLS 1.3 draft-10 PSK/PSK-(EC)DHE Handshake Protocol

Beyond the main, full handshake, TLS 1.3 specifies an abbreviated handshake based on pre-shared keys (PSKs) which, in its core structure, follows the main handshake structure, but omits some steps, especially the comparatively expensive signature-based authentication. Pre-shared keys may be established out-of-band or derived from the resumption master secret established in a full (EC)DHE handshake. TLS 1.3 specifies a pure pre-shared key–based mode (PSK) as well as a combined PSK and (ephemeral) Diffie–Hellman mode (PSK-(EC)DHE).

As the full handshake (cf. Section 6.2), the PSK-based handshakes are subdivided into the key exchange phase (including new `ClientPreSharedKey` and `ServerPreSharedKey` messages described below), the server parameters phase (reduced to server extensions), and the authentication phase (purely based on the `Finished` messages). Figure 6.2 shows the message flow and relevant cryptographic computations for the PSK and PSK-(EC)DHE handshake in `draft-10`. The new handshake message is the following:

- `ClientPreSharedKey` (`CPSK`)/`ServerPreSharedKey` (`SPSK`) are mandatory extensions for the PSK-based handshakes in which the client announces one (or multiple) pre-shared key identifier(s) (`psk_id`), of which the server selects one to be used as the pre-shared secret (`pss`) in the handshake.

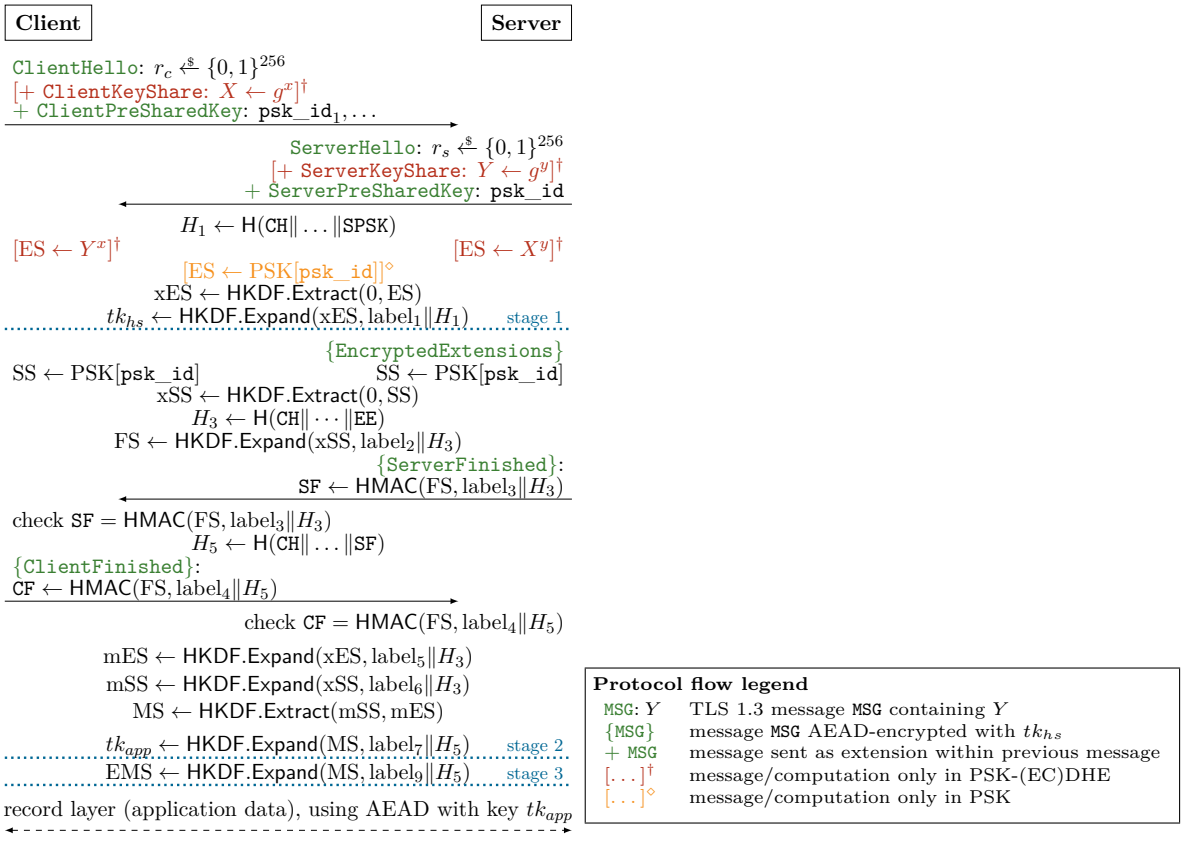


Figure 6.2: The TLS 1.3 `draft-10` PSK and PSK-(EC)DHE handshake protocol. The figural key schedule is identical to that of Figure 6.1, except that no RMS is derived.

The key schedule is almost identical to that of the full (EC)DHE handshake (see Figure 6.1), with the following three differences. First, in PSK(-only) mode the ephemeral secret ES is set to be the pre-shared secret pss = PSK[psk_id] agreed upon by the pre-shared key identifier psk_id. Second, the static secret SS in PSK modes is always the pre-shared secret pss = PSK[psk_id]. Third, the PSK-based handshakes do not derive a resumption master secret RMS, making the derivation of the exporter master secret EMS the third and last stage in the protocol.

6.5 Security of the TLS 1.3 draft-10 PSK/PSK-(EC)DHE Handshake

We use the pre-shared secret variant of our multi-stage key exchange model (MS-PSKE) to assess the security of both PSK and PSK-(EC)DHE handshake modes of TLS 1.3 `draft-10`, treating the pre-shared keys used to run the handshakes as the long-term secrets.

We begin by defining the session identifiers for the three stages (note that RMS is not derived in PSK and PSK-(EC)DHE handshakes) establishing the handshake traffic key tk_{hs}, the application traffic key tk_{app}, and the exporter master secret EMS. As in the (EC)DHE handshake, we let them contain the unencrypted messages sent and received excluding the

finished messages:

$$\begin{aligned}\mathsf{sid}_1 &= (\texttt{ClientHello}, \texttt{ClientKeyShare}^\dagger, \texttt{ClientPreSharedKey},\\ &\quad\ \texttt{ServerHello}, \texttt{ServerKeyShare}^\dagger, \texttt{ServerPreSharedKey})\\ \mathsf{sid}_2 &= (\texttt{ClientHello}, \texttt{ClientKeyShare}^\dagger, \texttt{ClientPreSharedKey},\\ &\quad\ \texttt{ServerHello}, \texttt{ServerKeyShare}^\dagger, \texttt{ServerPreSharedKey}, \texttt{EncryptedExtensions})\\ \mathsf{sid}_3 &= (\mathsf{sid}_2, \text{“EMS”}).\end{aligned}$$

Messages indicated with † are only included in the PSK-(EC)DHE handshake mode.

The contributive identifiers are incrementally set with each flow of messages sent and received by each party for stage 1. So $\mathsf{cid}_1 = (\texttt{ClientHello}, \texttt{ClientKeyShare}, \texttt{ClientPreSharedKey})$ after the client's first messages is extended to $\mathsf{cid}_1 = \mathsf{sid}_1$, and $\mathsf{cid}_2 = \mathsf{cid}_3 = \mathsf{sid}_1$, similarly to `draft-10-(EC)DHE`.

As the two modes provide distinct security guarantees, we discuss them separately below.

6.5.1 Security of the PSK Handshake

The TLS 1.3 `draft-10` PSK(-only) handshake (`draft-10-PSK`) provides key independence but no forward secrecy (as we will see, due to the lack of any ephemeral key material), along with the following protocol-specific properties $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$:

- $\mathsf{M} = 3$: `draft-10-PSK` consists of three stages deriving the handshake and application traffic keys tk_{hs} and tk_{app} and the exporter master secret EMS (in this order).
- $\mathsf{AUTH} = \{(\mathsf{mutual}, \mathsf{mutual}, \mathsf{mutual})\}$: all three derived keys are mutually authenticated (down to the pre-shared secret used).
- $\mathsf{USE} = (\mathsf{internal}, \mathsf{external}, \mathsf{external})$: The handshake traffic key is used internally within `draft-10-PSK` to encrypt the second part of the handshake; the two other keys are not used within the main handshake.
- $\mathsf{REPLAY} = (\mathsf{nonreplayable}, \mathsf{nonreplayable}, \mathsf{nonreplayable})$: All stages' keys are non-replayable due to nonces included from both sides.

As the `draft-10` PSK-based handshakes do not involve semi-static keys, we can omit the NewSemiStaticKey and RevealSemiStaticKey queries in the following.

The formal security results for the TLS 1.3 `draft-10` PSK handshake are as follows.

Theorem 6.4 (Match security of `draft-10-PSK`)**.** *The TLS 1.3* `draft-10` *PSK handshake is* Match*-secure with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. For any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq n_s^2 \cdot 2^{-|nonce|},$$

where n_s *is the maximum number of sessions and* $|nonce| = 256$ *is the bit-length of the nonces.*

Similar to the full handshake, Match security follows from the choice of session identifiers to include all unencrypted messages, in particular guaranteeing that partnered sessions agree on the same key, authenticity, and contributive identifiers. The given security bound is for identifier collisions due to two honest sessions colliding in their nonce.

Proof. We need to show the six properties of Match security (cf. Definition 4.1). Note that the only condition that is changed from the public-key (MSKE) to the pre-shared secret setting (MS-PSKE) is the fourth one, which now also requires agreement on the pre-shared secret identifier psid.

1. *Sessions with the same session identifier for some stage hold the same key at that stage.* The session identifier in each stage fixes all data entering the key derivation step and the pre-shared secret identifier psid. Hence, equal session identifiers imply that both parties compute the same intermediate and session keys.

2. *Sessions with the same session identifier for some stage agree on that stage's authentication level.*
   Since `draft-10-PSK` only specifies (mutual, mutual, mutual) as authentication as this is trivially true.

3. *Sessions with the same session identifier for some stage share the same contributive identifier.*
   This holds since the contributive identifier values cid_1, cid_2, and cid_3 are final and equal sid_1 contained in the respective session identifiers once the latter are set.

4. *Sessions are partnered with the intended (authenticated) participant and for mutual authentication share the same pre-shared secret index.*
   Honest sessions are assured of a peer's identity and key index as the pre-shared secret identifier psid is included within `ClientPreSharedKey` and `ServerPreSharedKey`, contained in all stages' session identifiers. Since each party knows the mapping of key index k and psid, a party can determine the peer identity via this mapping, and agreement on the session identifiers implies agreement on the partner identities.

5. *Session identifiers are distinct for different stages.*
   This holds trivially as the different stages' identifiers have unique length.

6. *At most two sessions have the same session identifier at any non-replayable stage.*
   Both client and server nonces are included in all stages' session identifier and thus the probability of three-fold colliding session identifiers is bound by the probability of two nonces colliding: $n_s^2 \cdot 2^{-|\text{nonce}|}$, where n_s is the maximum number of sessions and $|\text{nonce}| = 256$ is the nonces' bit-length. □

Theorem 6.5 (Multi-Stage security of `draft-10-PSK`)**.** *The TLS 1.3* `draft-10` *PSK handshake is* Multi-Stage*-secure in a key-independent and non-forward-secret manner with properties* (M, AUTH, USE, REPLAY) *given above. Formally, for any efficient adversary $\mathcal{A}$ against the* Multi-Stage *security there exist efficient algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_5$ such that*

$$\mathsf{Adv}^{\text{Multi-Stage},\mathcal{D}}_{\texttt{draft-10-PSK},\mathcal{A}} \leq 3n_s \cdot \Big(\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_p \cdot \big(\mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Extract},\mathcal{B}_2} + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_3} + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Extract},\mathcal{B}_4} + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_5} \big) \Big),$$

where n_s is the maximum number of sessions and n_p is the maximum number of pre-shared secrets.

Proof. As in the proof of Theorem 6.2 for the full handshake, we first restrict the adversary $\mathcal{A}$ to make only a single Test query, reducing its advantage by a factor at most $1/3n_s$ via a hybrid argument that also fixes the tested session label and stage i.

Game 0. This initial game equals the Multi-Stage game with a single Test query, so

$$\mathsf{Adv}^{G_0}_{\texttt{draft-10-PSK},\mathcal{A}} = \mathsf{Adv}^{\text{1-Multi-Stage}}_{\texttt{draft-10-PSK},\mathcal{A}}.$$

Game 1. In this game, the challenger aborts the game if any two honest sessions compute the same hash value for different inputs in any evaluation of the hash function H. We can break the collision resistance of H in case of this event by letting a reduction $\mathcal{B}_1$ output the two distinct input values to H. Hence:

$$\mathsf{Adv}^{G_0}_{\texttt{draft-10-PSK},\mathcal{A}} \leq \mathsf{Adv}^{G_1}_{\texttt{draft-10-PSK},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

Game 2. As the next step, we guess the pre-shared secret pss (among the n_p secrets established) that the tested session will use, and the challenger aborts the game if that guess was wrong. This reduces the adversary's advantage by a factor of at most $1/n_p$, thus

$$\mathsf{Adv}^{G_1}_{\texttt{draft-10-PSK},\mathcal{A}} \leq n_p \cdot \mathsf{Adv}^{G_2}_{\texttt{draft-10-PSK},\mathcal{A}}.$$

Let $\mathsf{pss}_{U,V,k}$ be the guessed pre-shared secret.

Game 3. We next replace the pseudorandom function HKDF.Extract in all evaluations using the tested session's pre-shared secret $\mathsf{pss}_{U,V,k}$ as key by a (lazy-sampled) random function. This in particular affects the derivation of both the extracted ephemeral static secrets $\mathrm{xES} = \mathrm{xSS}$ in the tested (and any potential partnered) session, which is replaced by a random value $\widetilde{\mathrm{xES}} = \widetilde{\mathrm{xSS}} \xleftarrow{\$} \{0,1\}^\lambda$.

We can bound the difference this step introduces in the advantage of $\mathcal{A}$ by the security of HKDF.Extract as a pseudorandom function. Notice here that for any successful adversary (which hence cannot invoke Corrupt on $\mathsf{pss}_{U,V,k}$ used in the tested session), the pre-shared key is an unknown and uniformly random value. Hence, the simulation is sound and we establish

$$\mathsf{Adv}^{G_2}_{\texttt{draft-10-PSK},\mathcal{A}} \leq \mathsf{Adv}^{G_3}_{\texttt{draft-10-PSK},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2}.$$

Game 4. We can now replace the HKDF.Expand applications in the tested and other sessions running on the same pre-shared key (and, hence, same $\widetilde{\mathrm{xES}}$ and $\widetilde{\mathrm{xSS}}$ values) with a random function. Thereby, we in particular replace the handshake traffic key tk_{hs}, the expanded ephemeral and static secrecy mES, mSS, and the finished secret FS in the tested (and any partnered) session by random values $\widetilde{tk_{hs}}, \widetilde{\mathrm{mES}}, \widetilde{\mathrm{mSS}}, \widetilde{\mathrm{FS}} \xleftarrow{\$} \{0,1\}^\lambda$.

These values are moreover independent of any value derived in a non-partnered session (which the adversary may reveal). The Expand evaluations include the (hashed) session identifiers and, due to Game 1, different session identifiers cannot collide on the same hash value. Any non-partnered session sharing the same pre-shared secret hence derives independent keys.

We can, as before, bound the advantage difference introduces by this step by the PRF security of HKDF.Expand and obtain

$$\mathsf{Adv}^{G_3}_{\texttt{draft-10-PSK},\mathcal{A}} \leq \mathsf{Adv}^{G_4}_{\texttt{draft-10-PSK},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3}.$$

Game 5. Randomness and independence of $\widetilde{\mathrm{mES}}$ in the tested session then allows us to replace the derived master secret by a random value $\widetilde{\mathrm{MS}} \xleftarrow{\$} \{0,1\}^\lambda$, a step which is again reducible to the PRF security of HKDF.Extract:

$$\mathsf{Adv}^{G_4}_{\texttt{draft-10-PSK},\mathcal{A}} \leq \mathsf{Adv}^{G_5}_{\texttt{draft-10-PSK},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_4}.$$

Game 6. As the last change, we can now replace the application traffic key tk_{app} and the exporter master secret EMS derived from $\widetilde{\mathsf{MS}}$ by random values, which is undetectable given the PRF security of HKDF.Expand:

$$\mathsf{Adv}^{G_5}_{\texttt{draft-10-PSK},\mathcal{A}} \leq \mathsf{Adv}^{G_6}_{\texttt{draft-10-PSK},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5}.$$

Finally, in Game 6 we reached a situation where all stages' keys in the tested session are chosen uniformly at random. Moreover, any non-partnered session (which $\mathcal{A}$ could potentially reveal) using the same pre-shared secret key material derives distinct session keys. Hence, $\mathcal{A}$ is left with no better chance then guessing:

$$\mathsf{Adv}^{G_6}_{\texttt{draft-10-PSK},\mathcal{A}} \leq 0.$$

Combining the given bounds yields the overall security statement. □

6.5.2 Security of the PSK-(EC)DHE Handshake

We now turn to the combined PSK and Diffie–Hellman mode, the TLS 1.3 `draft-10` PSK-(EC)DHE handshake (`draft-10-PSK-(EC)DHE`). As the PSK-only mode it provides key independence, but the ephemeral Diffie–Hellman values added enable forward secrecy (from the first stage on). The remaining protocol-specific properties $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ are as follows:

- $\mathsf{M} = 3$: `draft-10-PSK-(EC)DHE` also has three stages (for tk_{hs}, tk_{app} and EMS).
- $\mathsf{AUTH} = \{(\mathsf{unauth}, \mathsf{mutual}, \mathsf{mutual})\}$: the handshake traffic key tk_{hs} is unauthenticated, the other two keys are mutually authenticated (down to the pre-shared secret used).
- $\mathsf{USE} = (\mathsf{internal}, \mathsf{external}, \mathsf{external})$: The handshake traffic key is again used internally within `draft-10-PSK-(EC)DHE`; the two other keys are not used within the main handshake.
- $\mathsf{REPLAY} = (\mathsf{nonreplayable}, \mathsf{nonreplayable}, \mathsf{nonreplayable})$: All stages' keys are non-replayable due to nonces included from both sides.

As before, we can omit the NewSemiStaticKey and RevealSemiStaticKey queries as no semi-static keys are involved in the PSK modes.

Let us now state the formal security results for the TLS 1.3 `draft-10` PSK-(EC)DHE handshake.

Theorem 6.6 (Match security of `draft-10-PSK-(EC)DHE`)**.** *The TLS 1.3* `draft-10` *PSK-(EC)DHE handshake is* Match*-secure with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. For any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\texttt{draft-10-(EC)DHE},\mathcal{A}} \leq n_s^2 \cdot 1/q \cdot 2^{-|nonce|},$$

where n_s *is the maximum number of sessions,* q *is the group order, and* $|nonce| = 256$ *is the bit-length of the nonces.*

Match security follows similar to the result for `draft-10-PSK` (see Theorem 6.4). The only differences are due to the added Diffie–Hellman values g^x and g^y, which are agreed upon by matching session identifiers (hence yielding same keys) and lower the probability of three honest sessions having colliding session identifiers by a factor of the group order q. We hence do not repeat a full proof here.

Theorem 6.7 (Multi-Stage security of draft-10-PSK-(EC)DHE)**.** *The TLS 1.3* draft-10 *PSK-(EC)DHE handshake is* Multi-Stage*-secure in a key-independent and stage-1-forward-secret manner with properties* (M, AUTH, USE, REPLAY) *given above. Formally, for any efficient adversary $\mathcal{A}$ against the* Multi-Stage *security there exist efficient algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_8$ such that*

$$\begin{aligned}\mathsf{Adv}^{\text{Multi-Stage},\mathcal{D}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} \leq 3n_s \cdot \Big(& \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} \\ & + n_s \cdot n_p \cdot \Big(\mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2} + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3} + \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{HMAC},\mathcal{B}_4} \Big) \\ & + n_s \cdot \Big(\mathsf{Adv}^{\text{snPRF-ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_5} + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6} + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_7} + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8} \Big) \Big),\end{aligned}$$

where n_s is the maximum number of sessions and n_p is the maximum number of pre-shared secrets.

Proof. As in our previous proofs, we consider the case that the adversary $\mathcal{A}$ makes a single Test query, reducing the advantage of $\mathcal{A}$ by a factor of $1/3n_s$ (as RMS is not derived in pre-shared key modes). Additionally, we now know the session with label label that is to be tested in stage i. We proceed via the following sequence of games.

Game 0. The initial game equals the Multi-Stage game with a single Test query, yielding in combination with the initial hybrid step that

$$\mathsf{Adv}^{\text{Multi-Stage},\mathcal{D}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} \leq 3n_s \cdot \mathsf{Adv}^{G_0}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}}.$$

Game 1. We first of all let the challenger abort the game if any two honest sessions compute the same hash value for different inputs in any evaluation of the hash function H. As in the proofs for draft-10-(EC)DHE and draft-10-PSK we can bound the probability of such and abort by the advantage $\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}$ of an adversary $\mathcal{B}_1$ against the collision resistance of the hash function H:

$$\mathsf{Adv}^{G_0}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_1}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

From this point on, our analysis considers two disjoint cases:

A. The adversary tests a session without honest contributive partner in the first stage.[27]

B. The adversary tests a session with an honest contributive partner in the first stage.

Case A. Test Session without Partner

We first consider the case that the tested session is without honest contributive partner in the first stage. Since for draft-10-PSK-(EC)DHE the first stage is always unauthenticated, the adversary cannot test a session in the first stage without an honest contributive partner, this restricts our focus to Test queries in stage 2 and 3.

Game A.0. This initial game equals Game 1, with the single Test query issued to a session without honest contributive partner in stage 1. That is,

$$\mathsf{Adv}^{G_{A.0}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} = \mathsf{Adv}^{G_1,\text{test without partner}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}}.$$

[27] Note that this tested session may be a client or a server session.

Game A.1. In this game, the challenger aborts immediately if a session accepts in the second stage without an honest contributive partner in stage 1. Let $\mathsf{abort}_{acc}^{G_{A.1},\mathcal{A}}$ denote this event this occurs. Then

$$\left|\mathsf{Adv}^{G_{A.0}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} - \mathsf{Adv}^{G_{A.1}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}}\right| \leq \Pr[\mathsf{abort}_{acc}^{G_{A.1},\mathcal{A}}].$$

We can immediately bound $\mathsf{Adv}^{G_{A.1}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}}$. Throughout this proof case we assume that the Test query is directed to a session without honest contributed partner in stage 1. Because the authentication type of the protocol is $(\mathsf{unauth}, \mathsf{mutual}, \mathsf{mutual})$, the Test query can only be directed to stage-2 or stage-3 keys of that session. As Game A.1 is aborted when the first such session accepts (in stage 2), there is after all no moment in that game where a successful adversary could issue a Test query. Hence,

$$\mathsf{Adv}^{G_{A.1}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} \leq 0.$$

It remains to bound $\Pr[\mathsf{abort}_{acc}^{G_{A.1},\mathcal{A}}]$. We do so via a sequence of games that continues on from Game A.1.

Game A.2. In this game, the challenger guesses a session (from at most n_s sessions in the game) and aborts if the guessed session is not the first session which accepts in the second stage without an honest contributive partner in stage 1. If the challenger guesses correctly (which happens with probability at least $1/n_s$), then this game aborts at exactly the same time as the previous game:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.1},\mathcal{A}}] \leq n_s \cdot \Pr[\mathsf{abort}_{acc}^{G_{A.2},\mathcal{A}}].$$

Note that, in this game, the guessed session, which is the first stage-2 session that accepts without honest contributive partner in the first stage, could not have been issued any $\mathsf{Corrupt}$ query, nor could a $\mathsf{Corrupt}$ query have been issued to any other session sharing the same pre-shared secret. This is because sessions using that pre-shared secret do not continue execution once the secret is corrupted, and this session has accepted, so no $\mathsf{Corrupt}$ could have happened before it accepted in stage 2. Since the game terminates once stage 2 has accepted, no $\mathsf{Corrupt}$ query could have been issued after, either.

This allows us, in the following games, to replace the unexposed pre-shared secret pss in the guessed and all other sessions sharing the same pss value without being inconsistent or detectable with regards to the $\mathsf{Corrupt}$ query.

Game A.3. In this game we guess the pre-shared secret pss (among the n_p secrets established) that the guessed session will use, and the challenger aborts the game if that guess was wrong. This reduces the adversary's advantage by a factor of at most $1/n_p$, thus:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.2},\mathcal{A}}] \leq n_p \cdot \Pr[\mathsf{abort}_{acc}^{G_{A.3},\mathcal{A}}].$$

Let $\mathsf{pss}_{U,V,k}$ be the guessed pre-shared secret.

Game A.4. We next replace the pseudorandom function $\mathsf{HKDF.Extract}$ in all evaluations using the guessed session's pre-shared secret $\mathsf{pss}_{U,V,k}$ as key by a lazy-sampled random function. Beyond other sessions using the same pre-shared secret, this in particular affects the derivation of xSS in the guessed session, which is replaced with a random value $\widetilde{\mathrm{xSS}} \xleftarrow{\$} \{0,1\}^\lambda$.

We bound this difference in the advantage of $\mathcal{A}$ by the security of $\mathsf{HKDF.Extract}$ as a pseudorandom function, via a reduction $\mathcal{B}_2$ using its PRF oracle whenever $\mathsf{HKDF.Extract}(\cdot, \mathsf{pss}_{U,V,k})$ is to be evaluated. In the case of the oracle computing the function, the simulation equals Game A.3, but if it computes a random function, the simulation equals Game A.4. For any successful

adversary (note that a successful adversary by Games A.2 and A.3 cannot corrupt $\mathsf{pss}_{U,V,k}$, i.e., cannot issue $\mathsf{Corrupt}(\mathsf{label})$ queries where $\mathsf{label.pss} = \mathsf{pss}_{U,V,k}$) the pre-shared secret is uniformly random and unknown to $\mathcal{A}$, so the simulation is sound. Thus

$$\Pr[\mathsf{abort}_{acc}^{G_{A.3},\mathcal{A}}] \leq \Pr[\mathsf{abort}_{acc}^{G_{A.4},\mathcal{A}}] + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2}.$$

Game A.5. In this step we replace the evaluations of $\mathsf{HKDF.Expand}$ using $\widetilde{\mathrm{xSS}}$ as key in the guessed session by a lazy-sampled random function, thereby exchanging the finished secret value FS and the expanded static secret mSS with independent random values $\widetilde{\mathrm{FS}}, \widetilde{\mathrm{mSS}} \xleftarrow{\$} \{0,1\}^\lambda$. We can bound this difference in the same manner as above, and thus:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.4},\mathcal{A}}] \leq \Pr[\mathsf{abort}_{acc}^{G_{A.5},\mathcal{A}}] + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3}.$$

Finally, we show how any adversary that manages to make the $\mathsf{abort}_{acc}^{G_{A.5},\mathcal{A}}$ event happen can be transformed into an adversary $\mathcal{B}_4$ that breaks the existential unforgeability of the HMAC scheme.

To this extent, let $\mathcal{B}_4$ simulate Game A.5 for $\mathcal{A}$ as specified, but when the guessed session or partner session requires a MAC computation using $\widetilde{\mathrm{FS}}$, $\mathcal{B}_4$ invokes its MAC oracle to generate that value. Since $\widetilde{\mathrm{FS}}$ is uniformly random and independent of all other values in the game, this simulation is sound.

Assume now $\mathcal{A}$ triggers $\mathsf{abort}_{acc}^{G_{A.5},\mathcal{A}}$. In this case, the accepting session must have received a `ServerFinished` (respectively, `ClientFinished`) message (when $\mathsf{role} = \mathsf{initiator}$, resp. $\mathsf{responder}$) that is a valid MAC tag over the session hash $H_3 = \mathsf{H}(\mathtt{CH}, ..., \mathtt{EE})$, resp. $H_5 = \mathsf{H}(\mathtt{CH}, ..., \mathtt{SF})$. Since every other honest session holds a different session identifier (as there exists no honest contributive partner in the first stage of the accepting session), no honest party will have issued a MAC tag on that session hash. Moreover, there exist no hash collisions by Game 1, so the MAC input is distinct to any other MAC input for any honest party. Therefore, this message was never queried to the MAC oracle and hence constitutes a MAC forgery. This allows us to conclusively bound the probability for abortion due to a stage-2 accepting session without stage-1 contributive identifier by

$$\Pr[\mathsf{abort}_{acc}^{G_{A.5},\mathcal{A}}] \leq \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_4}.$$

Summing the probabilities accumulated over the sequence of games, we obtain the bound for Case A:

$$\mathsf{Adv}^{G_1,\text{test without partner}}_{\mathtt{draft\text{-}10\text{-}PSK\text{-}(EC)DHE},\mathcal{A}} \leq n_s \cdot n_p \cdot \left(\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_4}\right).$$

Case B. Test Session with Partner

We now come to the case where the tested session has an honest contributive partner in the first stage.

Game B.0. This initial game equals Game 1, with the single Test query issued to a session having an honest contributive partner in stage 1. Thus,

$$\mathsf{Adv}^{G_{B.0}}_{\mathtt{draft\text{-}10\text{-}PSK\text{-}(EC)DHE},\mathcal{A}} = \mathsf{Adv}^{G_1,\text{test with partner}}_{\mathtt{draft\text{-}10\text{-}PSK\text{-}(EC)DHE},\mathcal{A}}.$$

Game B.1. Our first modification is to guess a session (from the at most n_s sessions in the game) and abort if the session guessed is not the honest contributive partner in stage 1 of the tested session. This reduces the adversary's advantage by a factor of at most $1/n_s$.

$$\mathsf{Adv}^{G_{B.0}}_{\mathtt{draft\text{-}10\text{-}PSK\text{-}(EC)DHE},\mathcal{A}} \leq n_s \cdot \mathsf{Adv}^{G_{B.1}}_{\mathtt{draft\text{-}10\text{-}PSK\text{-}(EC)DHE},\mathcal{A}}.$$

Game B.2. In this game, we replace the extracted ephemeral secret xES derived in the tested and (potentially) its contributive partner session with a uniformly random and independent string $\widetilde{\text{xES}} \xleftarrow{\$} \{0,1\}^\lambda$. As in Game B.2 of the proof for `draft-10-(EC)DHE` (cf. Theorem 6.2 in Section 6.3), we employ the snPRF-ODH assumption in order to be able to simulate the computation of xES in a partnered client session for a modified `ServerKeyShare` message when encoding a DDH challenge tuple.

More precisely, we can turn any adversary capable of distinguishing the change in this game into an adversary $\mathcal{B}_5$ against the snPRF-ODH security of the HKDF.Extract function (keyed with ES on label 0). For this, $\mathcal{B}_5$ asks for a PRF challenge on 0. It uses the obtained Diffie–Hellman shares g^x and g^y within `ClientKeyShare` and `ServerKeyShare`, respectively, of the tested and contributive partner session, and the PRF challenge value as xES in the test session. If necessary, $\mathcal{B}_5$ uses its single ODH_u query in the snPRF-ODH game to derive xES in the partnered session on differing $g^{y'} \neq g^y$.

Providing a sound simulation of either Game B.1 (if the PRF challenge value is real) or Game B.2 (if the PRF challenge value is random), this bounds the advantage difference of $\mathcal{A}$ as

$$\mathsf{Adv}^{G_{B.1}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.2}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_5}.$$

Game B.3. In this game, we replace the handshake traffic key tk_{hs} and the expanded ephemeral secret mES derived in both the tested and its contributive partner session with a uniformly random and independent strings $\widetilde{tk_{hs}}$, $\widetilde{\text{mES}} \xleftarrow{\$} \{0,1\}^\lambda$ in the tested and partner session. We can turn any adversary capable of distinguishing this change into an adversary $\mathcal{B}_6$ against the PRF security of the HKDF.Expand function keyed with $\widetilde{\text{xES}}$. We let $\mathcal{B}_6$ simulate the previous game as the challenger, except that it queries its PRF oracle for the derivation of tk_{hs} and mES from $\widetilde{\text{xES}}$. If the oracle computes the PRF, we are in Game B.2, but if it computes a random function, we are in Game B.3 as $\widetilde{\text{xES}}$ is uniformly random and independent bit string. The advantage of $\mathcal{B}_6$ in the PRF security game bounds the advantage of this change, such that

$$\mathsf{Adv}^{G_{B.2}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.3}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6}.$$

Game B.4. In this game, we replace the master secret MS derived from $\widetilde{\text{mES}}$ in both the tested and its contributive partner session with a uniformly random and independent string $\widetilde{\text{MS}} \xleftarrow{\$} \{0,1\}^\lambda$ in the tested and partner session. We can turn any adversary capable of distinguishing this change into an adversary $\mathcal{B}_7$ against the security of the HKDF.Extract function which we again model as a pseudorandom function. Via a similar argument to the previous games we find

$$\mathsf{Adv}^{G_{B.3}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.4}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_7}.$$

Game B.5. In this game, we replace the application traffic key tk_{app} and the exporter master secret EMS derived from $\widetilde{\text{MS}}$ in both the tested and its contributive partner session with a uniformly random and independent strings $\widetilde{tk_{app}}, \widetilde{\text{EMS}} \xleftarrow{\$} \{0,1\}^\lambda$ in the tested and partner session. We can turn any adversary capable of distinguishing this change into an adversary $\mathcal{B}_8$ against the security of the HKDF.Expand function which we still model as a pseudorandom function. Via a similar argument as the previous games we find

$$\mathsf{Adv}^{G_{B.4}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.5}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8}.$$

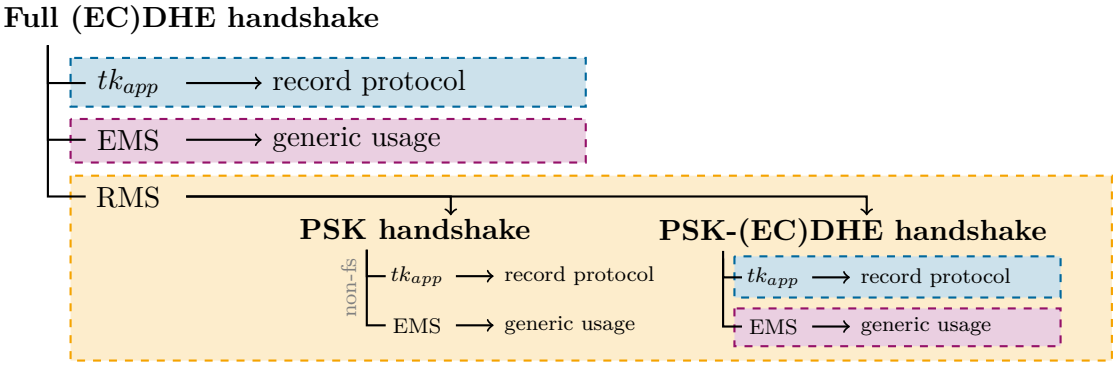


Figure 6.3: Illustration of the compositional guarantees for external keys in the full (EC)DHE and PSK-based handshakes of TLS 1.3 `draft-10`. Derived keys are connected to the handshake by solid lines, their usage in protocols is indicated by an arrow. Dashed boxes indicate an application of the composition result (Theorem 4.4) to the usage of a specific key in a subsequent symmetric-key protocol.

Note that now $\widetilde{tk_{hs}}$, $\widetilde{tk_{app}}$ and $\widetilde{\text{EMS}}$ are uniformly random bit strings independent of all other values. Moreover, any non-partnered session (that the adversary might reveal) derives different keys, as the distinct (hashed) session identifier enters the key derivation for each stage's key. The response to the Test query therefore is now independent of the test bit b_{test}, so $\mathcal{A}$ cannot distinguish the real from the random case and thus

$$\mathsf{Adv}^{G_{B.5}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} \leq 0.$$

This yields the following security bound for Case B:

$$\mathsf{Adv}^{\text{1-Multi-Stage,session with partner}}_{\texttt{draft-10-PSK-(EC)DHE},\mathcal{A}} \leq n_s \cdot \Big(\mathsf{Adv}^{\mathsf{snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_5} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_7} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8}\Big). \qquad \square$$

6.6 Composition

Leveraging our composition theorem established for the multi-stage key exchange model in Section 4.6, we can now establish compositional guarantees for the external keys of both the full (EC)DHE and the PSK-based handshakes of TLS 1.3 `draft-10`, illustrated in Figure 6.3.

Recall that for the generic composition theorem to apply to some stage i key, we need that the key exchange protocol is $\mathsf{Multi\text{-}Stage}$-secure providing key independence, stage-j forward secrecy for $j \leq i$, multi-stage session matching, and the stage i keys to be external and non-replayable (cf. Theorem 4.4). It is easy to see that all handshake modes have multi-stage session matching, as the session identifier for any stage can always be computed from the public transcript and the keys of previous stages. We established the remaining conditions for the external keys, i.e., the application traffic key tk_{app}, the exporter master secret EMS, and the resumption master secret RMS in the $\mathsf{Multi\text{-}Stage}$ security results for the (EC)DHE and PSK-(EC)DHE handshake (cf. Theorems 6.2 and 6.7). Note that keys derived in a PSK-only handshake do not meet the conditions for composition as this handshake mode lacks forward secrecy.

The compositional result hence supports the security of using the application traffic key in the TLS 1.3 record protocol (for which our result enables an independent analysis) and the generic use of the exporter master secret. Moreover, it allows us to cascade the security analysis of the PSK-based handshakes using the resumption master secret of a full handshake as pre-shared key, treating our pre-shared secret multi-stage key exchange security model (MS-PSKE) as an instance of a generic symmetric-key protocol.

6.7 Comments on the TLS 1.3 Handshake Design

As part of our analysis effort in parallel to the development of the TLS 1.3 draft standards we provided comments to the community and in particular the IETF TLS working group, e.g., through its mailing list [Gün15], workshops on the status of TLS 1.3 [TRO16a, TRO16b, TLS17], or direct contribution to the RFC draft [Res18]. Overall and as established in this chapter, TLS 1.3 (in `draft-10`) achieves its main cryptographic goals, including session-key indistinguishability and independence as well as privacy of (the key used for) encrypted handshake messages. In the following, we furthermore note aspects that enable or facilitate our proofs as well as comments on some cryptographic choices in the TLS 1.3 draft handshakes. We focus here on the full (EC)DHE and PSK-based handshakes and remark that, while some notes are informed by changes in later draft versions, our comments mainly refer to common design aspects shared by `draft-10`. We provide further comments and discussion on the low-latency 0-RTT handshake modes (both Diffie–Hellman- and PSK-based) in Chapter 7, specifically on the handling of replays in Section 7.6.

Soundness of key separation. Earlier versions of TLS used the same session key to encrypt application data as well as the `Finished` messages at the end of the handshake. This made it impossible to show that the TLS session key satisfied standard Bellare–Rogaway-style key indistinguishability security [BR94]. We confirm that the change in keys for encryption of handshake messages allows the TLS 1.3 drafts to achieve standard key indistinguishability security.

Key independence. The discussed TLS 1.3 drafts achieve key independence in the multi-stage security setting, which heavily strengthens their overall security. (Recall key independence is the property that one can reveal a session key without endangering the security of later-stage keys.) Beyond making it amenable to generic composition, key independence safeguards the usage of derived keys against inter-protocol effects of security breakdowns.

In particular, deriving separate resumption and exporter master secret (beyond the application traffic key) enhances modularity and composability: exported key material to arbitrary external applications is fully detached from the main master secret, which is in contrast to previous TLS versions where the exporting processes requires separate cryptographic treatment (e.g., [BJS16]).

Signing the session hash. In the TLS 1.3 full handshake, authenticating parties (the server, and sometimes the client) sign (the hash of) all handshake messages up to when the signature is issued. This is different from TLS 1.2 and earlier, where the server's signature is only over the client and server random nonces and the server's ephemeral public key. Signing the transcript for authentication facilitates our proofs of the protocol's authentication properties.

Encryption of handshake messages. The TLS 1.3 drafts encrypt the second part of the handshake using the initial handshake traffic key tk_{hs}, aiming to provide some form of privacy (against passive adversaries) for these messages, in particular for the server and client certificates. Our analysis shows that the handshake traffic key does indeed have security against passive adversaries and hence increases the handshake's privacy. The secrecy of the final session keys however does not rely on the handshake being encrypted and would remain secure even if it was done in clear. Our analysis considers the encrypted case, showing that this encryption does not negatively affect the security goals.

Finished messages. The `Finished` messages in both drafts are computed as an HMAC MAC value to the (hash of the) handshake transcript. Interestingly, in our proof for `draft-10` the `Finished` messages do not contribute to the session key secrecy in the full handshake in the sense that the key exchange would be secure without these messages. This is mainly because the signatures already authenticate the transcript. This contrasts with the case of RSA key transport in the TLS 1.2 full handshake: the analyses of both Krawczyk et al. [KPW13] and Bhargavan et al. [BFK+14] note potential weaknesses or require stronger security assumptions if `Finished` messages are omitted. From an engineering perspective, the `Finished` messages may still be interpreted as providing some form of (explicit) session key confirmation (cf. Section 8.3), and for PSK-based handshakes they provide the only form of (explicit) authentication.

Notably, with the introduction of a 0.5-RTT communication option (where the server can send data to the client along with its handshake flight), deriving the application traffic key from a truncated transcript makes the (existential unforgeability of the) `Finished` MACs a necessary component for key secrecy again. We will see and discuss this further in our analysis of the Diffie–Hellman-based 0-RTT handshake for TLS 1.3 `draft-12` in Section 7.5.

Session hash in key derivation. The TLS 1.3 drafts include a hash of all messages exchanged so far in the derivation of most session key (but see the discussion on truncated transcripts in the derivation of (early) application keys from `draft-12` on, Section 7.5). This session hash was introduced in response to the triple handshake attack [BDF+14] on TLS 1.2 and earlier. Our analyses confirm its usefulness in ensuring that sessions with different session identifiers have different master secrets and session keys.

Upstream hashing in signatures, MACs, and key derivation. In signing (resp. MACing) the transcript for authentication as well as in deriving keys via HKDF, TLS 1.3 uses the *hash* of the current transcript as input; if, e.g., the signature algorithm is a hash-then-sign algorithm, it will then perform an additional hash. From a cryptographic point of view, it would be preferable to insert the full (unhashed) transcript and let the respective signature, MAC, or KDF algorithms opaquely take care of processing this message. For engineering purposes, however, it may be desirable to hash the transcript iteratively, only storing the intermediate values instead of entire transcript. In our security proof, this upstream hashing leads to an additional assumption about the collision resistance of the hash function (which would otherwise be taken care of by the signature, MAC, resp. KDF scheme).

Chapter 7

The TLS 1.3 Protocol: Zero Round-Trip Time and Replays

Summary. In this chapter we present our security analysis of the low-latency zero round-trip time (0-RTT) handshake mode candidates of TLS 1.3, both in its (deprecated) Diffie–Hellman-based variant from version `draft-12` [Res16a] and its (remaining) pre-shared key–based variant from `draft-14` [Res16c]. We begin by discussing general concepts for 0-RTT key establishment and the effects of only one side actively contributing to the derived key on the security level achievable. A particular focus is on replay protection and which kind of guarantees can (not) be expected. We then describe the TLS 1.3 0-RTT handshake candidates with their differences in cryptographic operations compared to the standard Diffie–Hellman and PSK handshakes. Subsequently, we give our security results in the multi-stage key exchange model, reflecting the reduced replay protection guarantees. We furthermore compare the 0-RTT key exchange design of TLS 1.3 with that of Google's QUIC protocol. The results in this chapter are based on a work published at IEEE EuroS&P 2017 [FG17].

7.1 Introduction

While efficiency has always been a relevant aspect for key exchange protocols, optimization traditionally focused on cryptographic operations which for a long time dominated the overall cost (in time) for executions. With the technological progress in speed of computation, but also advances and, equally important, the deployment of elliptic-curve cryptography, researchers and practitioners managed to reduce the cost of (even asymmetric) cryptographic operations drastically over the last decades. As a result, the communication complexity has become a more and more dominant factor for the overall efficiency of key exchange protocols.

7.1.1 Zero Round-Trip Time

While steadily increasing bandwidth on the Internet renders the data complexity aspect of communication subordinate, speed of light prepares to set a definitive lower bound for the time a message needs to be sent back and forth between two parties (called *round-trip time*). Reducing the round complexity has hence become a major design criteria in the last years, with several low-latency designs for key exchange proposed by researchers [PZS+13, KW16, WTSB16, HJLS17] as well as by practitioners. Prominent practical examples are in particular Google's QUIC protocol [QUI] incorporated into the Chrome browser (see also Chapter 5) and the upcoming TLS version 1.3 [Res18], the latter being based on the OPTLS key exchange protocol by Krawczyk and Wee [KW16]. Those designs set out to establish an initial key in *zero round-trip*

time (0-RTT) that allows one party (usually the client) to send "early" data already along with the first key exchange message to a (previously visited) server.

Without the server being able to contribute, it is well understood that such an approach cannot achieve equally strong security guarantees for the initial key as classical key exchange protocols are able to provide with a full round-trip (and hence contributions from both parties). In particular, the initial key cannot provide (forward) secrecy in a setting where no state is shared between sessions and all but the ephemeral keying material is compromised after the key exchange run.[28] The common strategy is, hence, that both parties switch to a stronger key (e.g., achieving forward secrecy) after the server contributed in a second step of the key exchange and protect any further communication under this key.

Diffie–Hellman-based 0-RTT. One main concept to derive a 0-RTT key based on a Diffie–Hellman-style key exchange and to later upgrade to a stronger, forward-secret key, is shared by both recent prominent instances QUIC and TLS 1.3 (up to `draft-12` [Res16a]).[29] From a high-level perspective (i.e., omitting necessary mechanisms to protect, e.g., against replays or man-in-the-middle attacks which both protocols employ), this concept works as follows. Prior to the actual key exchange, the client is assumed to have talked to the server before and, in that communication, obtained a so-called *server configuration*. Cryptographically speaking, this configuration includes a *semi-static* Diffie–Hellman share g^s, for which the server stores the secret exponent s for a certain time. In QUIC, authentication of this server configuration is via an (offline) signed structure announced by the server, in TLS 1.3 it is signed (online) during a prior handshake.

Within its first message in subsequent executions, the client then sends an ephemeral Diffie–Hellman share g^x, derives the 0-RTT key key_1 as (a function of) $(g^s)^x = g^{xs}$, and is hence immediately able to, e.g., send encrypted data under the key. The server then computes the same key as $(g^x)^s$ (enabling decryption of the 0-RTT data) and responds with its own ephemeral share g^y for the stronger shared key. Both parties derive a second key key_2 as (a function of) g^{xy}, which can then enjoy forward secrecy in the sense that it remains secure even if g^s or the parties long-term secrets are later compromised.

Pre-shared key–based 0-RTT. Another concept for establishing a key in zero round-trip time is based on pre-shared keys (PSKs) and, from `draft-13` [Res16b] on, forms the basis of the only 0-RTT handshake mode specified for TLS 1.3 (i.e., the option for Diffie–Hellman-based 0-RTT was deferred in `draft-13`). Here, the 0-RTT key key_1 is derived from a previously established secret key (e.g., in TLS 1.3 the resumption master secret established in a regular handshake). The client can perform this computation without interaction with the server and hence is able to immediately send encrypted data under key_1. Later, both parties update to a second key key_2 derived from the pre-shared secret and possibly further exchanged material, e.g., fresh Diffie–Hellman shares to ensure forward secrecy.

7.1.2 The Problem with Replays and How It Is (Not) Solved in QUIC and TLS 1.3

The standard approach in key exchange protocols to prevent a man-in-the-middle attacker from replaying messages in order to make a party derive the same key twice is to include a nonce in both the client's and the server's messages and let the nonce contribute to the derived key. For

[28] We remark that forward secrecy is achievable in 0-RTT when sessions can share state, as we show in our work on forward-secret 0-RTT key exchange based on (forward-secret) puncturable encryption [GHJL17].

[29] We refer here to the (EC)DHE 0-RTT variant in TLS 1.3 draft `draft-12` [Res16a] and the original QUIC proposal in Revision 20130620 [LC13].

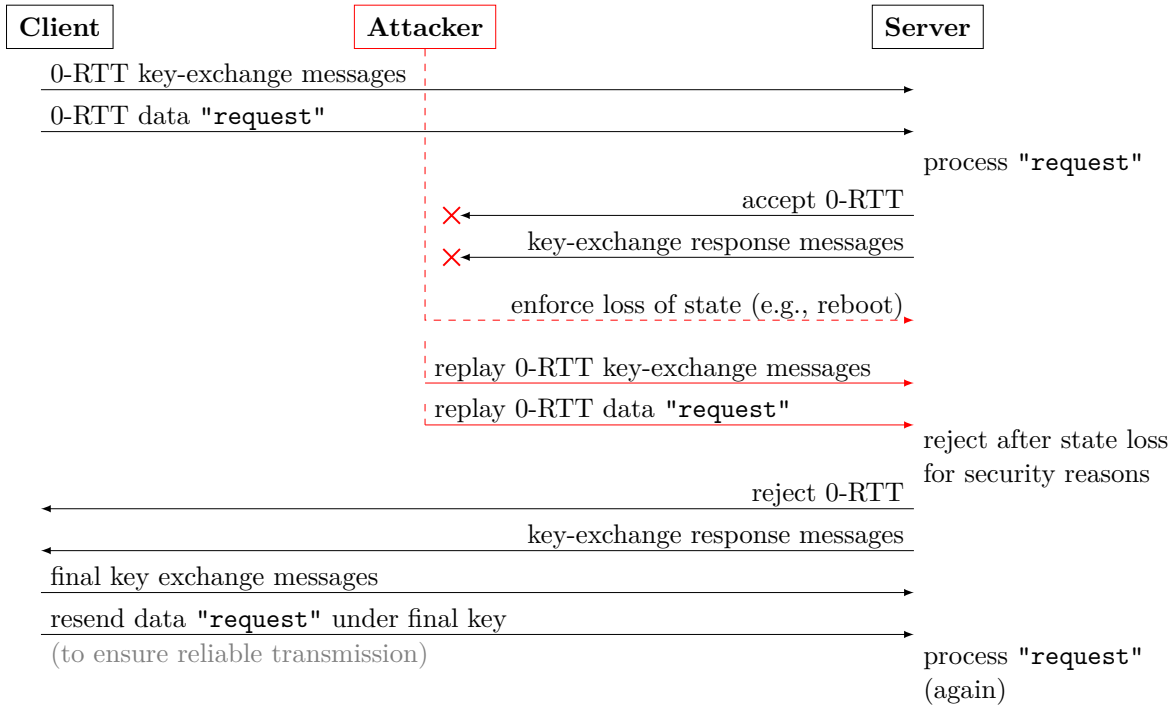


Figure 7.1: Generic replay attack discovered by Daniel Kahn Gillmor in the IETF TLS working group discussion around TLS 1.3 [Res15a]. The 0-RTT data `"request"` could, e.g., be an HTTP request `"POST /buy-something"`.

a 0-RTT key exchange, which is essentially a one-pass (i.e., one-message) key exchange protocol [BWM99a], messages (and hence keys) are—at first glance—inevitably replayable[30].

The QUIC protocol side-stepped the replay problem in its original cryptographic design [LC13] (Revision 20130620) by demanding the server to store all nonces seen in a so-called "strike register"—restricted in size by a server-specific "orbit" prefix and current time contained in the nonces—and rejecting any recurring nonce. As our security analysis in Chapter 5 as well as the one by Lychev et al. [LJBN15] confirmed, this approach indeed allows to establish a secure 0-RTT key which is non-replayable in the sense that no adversary can make a party derive the same key twice. However, while this approach succeeds to prevent *replays on the key-exchange level* (in terms of preventing double-derivation of keys), it is not sufficient to prevent *(logical) replays of the actual data exchanged*, in particular when it comes to real-world settings where a server entity is implemented in a cluster of (potentially distributed) servers, as we explain next. Let us stress that this problem with replays is independent of whether the 0-RTT key exchange is based on Diffie–Hellman or on pre-shared keys.

As discovered by Daniel Kahn Gillmor in the discussion around the upcoming TLS version 1.3 [Res15a] (see also the discussion in the TLS 1.3 draft [Res18, Section 8]), any 0-RTT anti-replay mechanism deployed at the key exchange level may become void when combined within an overall channel protocol that aims to provide reliable delivery of data messages (like, e.g., QUIC or TLS). The reason is that such a protocol may automatically resend rejected 0-RTT data under the second (final) key derived in order to ensure delivery. A generic attacker can then, for any client sending 0-RTT key-exchange messages together with some encrypted data, make this data be delivered twice in the following attack, also illustrated in Figure 7.1

[30] We use the notion of replays interchangeably for both messages and the keys computed based on those replayed messages.

(see [Res15a] for a more detailed description of the attack).

The attacker first conveys the client's 0-RTT messages and encrypted data to the server (which processes it), but drops the server's key exchange response. It then forces the server to lose its state, e.g., through rebooting, and presents the same messages again to the server. The server, with knowledge about its reset, has to conservatively decline the 0-RTT part of the key exchange for security reasons, but will reply with its own key exchange contribution for the final key which the attacker now simply relays to the client. The client derives the final key and, to ensure reliable delivery, *sends the desired data again* under this key, which the server will hence decrypt and process a second time. This constitutes a replay of the contained application data and might, e.g., result in a web transaction being processed twice.

Note that the contrived requirement that the attacker is able to reboot the server (while the client keeps waiting for a response) vanishes in a real-world scenario with distributed server clusters, where the attacker instead simply forwards the 0-RTT messages to two servers and drops the first server's response. The described attack hence in particular affects the cryptographic design of QUIC, which (among others) specifically targets settings with distributed clusters. Holding up the originally envisioned 0-RTT full replay protection being impossible, Langley and Chang write in the specification of December 2016 [LC16] (Revision 20161206) that this design is "destined to die" and will be replaced by (an adapted version of) the TLS 1.3 handshake (see also [LRW+17, TT17]). We, however, argue here that QUIC's strategy in Revision 20130620 still supports some kind of replay resistance, only at a different level. In contrast, TLS 1.3 in principle forgoes cryptographic protection mechanisms and instead accepts replays as inevitable (on the channel level). Yet, recent changes in the latest TLS 1.3 drafts [Res18, Section 8] re-added certain optional anti-replay techniques as recommended based on discussions on 0-RTT security in applications [Mac17], supporting our argument in favor of key-exchange–level replay protection.

There is, then, a significant conceptual gap between replays (of key-exchange messages and keys) on the key-exchange level, and the replay of user data faced on the level of the overall secure channel protocol in the 0-RTT setting. While the former can effectively be prevented within the key exchange protocol, this does not necessarily prevent the latter which can be (and in practice is) induced by the network stack of the channel actively and automatically re-sending (presumably) rejected 0-RTT data under the main key. The latter type of logical, network-stack replays is hence fundamentally beyond of what key exchange protocols can protect against.

7.1.3 0-RTT Security in TLS 1.3 and a Comparison with QUIC

In this chapter, we take a look at the 0-RTT handshake modes of TLS 1.3 and provide a comparison with the QUIC design, especially with respect to the issues with replays. We analyze both the pre-shared key–based (PSK-based) 0-RTT handshake mode from TLS 1.3 `draft-14` [Res16c] as well as the previously abandoned Diffie–Hellman-based (DH-based) 0-RTT handshake mode in its last specified form in `draft-12` [Res16a]. While doing so, we discuss the commonalities and differences between (the original version of) QUIC and the two TLS 1.3 modes in this respect.

As mentioned, the Diffie–Hellman-based ((EC)DHE) 0-RTT handshake was removed from the TLS 1.3 draft specification in `draft-13`, leaving only a PSK-based 0-RTT mode (with or without additional Diffie–Hellman exchange) in the latest drafts. Still, the (EC)DHE 0-RTT variant is much closer to the QUIC and OPTLS proposals, and it may be used as a TLS extension [Res16e], especially since it provides some kind of forward secrecy [Kra16a]. We hence also provide an analysis of the DH-based variant to enable a comparison of the security guarantees provided, but focus on the PSK-based 0-RTT handshake specified for draft `draft-14`.

Multi-stage key exchange with replayable 0-RTT keys. As the original QUIC key exchange Revision 20130620 [LC13] ensures non-replayability (on the key-exchange level), our analysis in Chapter 5 in the multi-stage setting did not need to consider replays. The TLS 1.3 0-RTT handshake candidates (both `draft-12` DH-based and `draft-14` PSK-based) however, as discussed, do not include any cryptographic anti-replay mechanisms even on the key-exchange level.

We hence now make use of the replayability components introduced with the multi-stage key exchange model in Chapter 4. The distinction between replayable and non-replayable stages (and, hence, keys) allows us to capture that the TLS 1.3 0-RTT key exchange messages of a client session can be replayed to multiple server sessions which will all derive the same key. In order to capture the effects of exposures of the semi-static keys used in a DH-based handshake to non-interactively derive the 0-RTT keys, the model allows them to be compromised and defines how this affects both 0-RTT keys (which will be compromised) and non–0-RTT keys (which are required to remain secure).

Security analysis of the TLS 1.3 draft-14 PSK/PSK-(EC)DHE 0-RTT and draft-12 (EC)DHE 0-RTT handshakes. We apply our model (in Sections 7.3 and 7.5) to analyze the PSK and PSK-(EC)DHE 0-RTT handshake modes specified in TLS 1.3 `draft-14`, which we describe in Section 7.2 first, as well as the (EC)DHE 0-RTT handshake mode of `draft-12`, described in Section 7.4. For the analysis of the other specified handshake modes of TLS 1.3, full (EC)DHE and pre-shared key, in the (relatively close) `draft-10` see Chapter 6. Our analysis shows that all three 0-RTT handshakes are secure (multi-stage) key exchange protocols, establishing random-looking keys. In particular, the two 0-RTT keys derived to protect the early handshake messages and application data, tk_{ehs} resp. tk_{eapp}, achieve the desired unilateral resp. mutual authentication, and are—as expected—replayable. Furthermore, we confirm that the second parts of the handshakes (essentially a full (EC)DHE resp. a regular PSK/PSK-(EC)DHE handshake), achieve security similar to that when run without 0-RTT prelude in `draft-10`. Our model allows us to show that security holds for the different authentication options of TLS 1.3 running in parallel and that all keys derived are independent in the sense of that leaking one of them does not affect any other key.

Our security results hold under mostly standard cryptographic assumptions like the unforgeability of the signature and MAC scheme, collision resistance of the hash function, and pseudorandomness properties of the key derivation function. The handshakes' security further relies on the pseudorandom-function oracle Diffie–Hellman (PRF-ODH) assumption (in its snPRF-ODH variant), introduced and used earlier for the analysis of several Diffie–Hellman–based modes of TLS 1.2 [JKSS12, KPW13]. Notably, for technical reasons that we detail in our proof, we furthermore need to employ a slightly strengthened, double-sided variant of the PRF-ODH assumption which we defined under the name of msPRF-ODH in Definition 3.6 (Section 3.2.2) for the analysis of the (EC)DHE 0-RTT handshake (in `draft-12`); we refer to our study of the PRF-ODH assumption for a detailed comparison of its different variants [BFGJ17].

Comparison of QUIC and TLS 1.3. We point out, in passing and explicitly in Section 7.6, how the designs of QUIC and TLS 1.3 differ in the way of handling 0-RTT, replay attacks, and data, and how this affects the security. This testifies that, although there may be an agreement on the general goals which should be achieved with 0-RTT, the technical details can vary significantly. One major difference has already been discussed above, carving out that both protocols treat replays differently. Another difference is that QUIC basically restarts the key exchange for an invalid (rejected) 0-RTT request, whereas TLS 1.3 instead only skips over to the regular handshake part. Both protocols also employ different approaches to derive the

session keys: While QUIC uses the early key for transmitting both key-exchange messages and application data, TLS 1.3 uses a more versatile approach to create early keys for designated purposes and thus achieves stronger security guarantees and better modularity.

7.2 The TLS 1.3 draft-14 PSK and PSK-(EC)DHE 0-RTT Handshake Protocols

Starting from `draft-13`, TLS 1.3 only specifies PSK-based 0-RTT handshake modes, abandoning the (EC)DHE-based variant predominant in earlier drafts. We hence first present our analysis of the pre-shared key–based variants, PSK(-only) 0-RTT and PSK-(EC)DHE 0-RTT, as specified in TLS 1.3 `draft-14` [Res16c]. In the PSK 0-RTT mode, keys are solely derived from a beforehand established pre-shared key (usually the resumption master secret RMS derived in a full TLS 1.3 handshake, cf. Section 6.2). In the PSK-(EC)DHE 0-RTT mode, (elliptic curve) Diffie–Hellman shares additionally enter the key derivation.

Compared to the regular PSK-based TLS 1.3 handshake (cf. Section 6.4), the PSK-based 0-RTT handshakes are augmented with an additional conceptual phase, now comprising the following four phases:

Key exchange. In the key exchange phase, parties as before negotiate the ciphersuites and key-exchange parameters to be used and establish shared key material as well as traffic keys to encrypt the remaining handshake.

0-RTT. In the added 0-RTT (data) phase, which is interleaved with the key exchange phase, the client can send application data already in its first flight. For this purpose, traffic keys for encrypting the early handshake and application data are established.[31]

Server parameters. In the server parameters phase, further handshake parameters (as, e.g., whether client authentication is demanded) are fixed by the server, as before.

Authentication. In the authentication phase, both the server and client can (based on the aspired authentication) authenticate, verify that they share the same view of the handshake, and derive (authenticated) application traffic keys. This phase is also like in the regular handshake.

We illustrate the protocol flow (with the cryptographically relevant computations) and the key schedule for both the PSK(-only) and PSK-(EC)DHE 0-RTT handshakes in Figure 7.2. The handshake messages `ClientHello`, `ServerHello`, `ClientPreSharedKey`, `ServerPreSharedKey`, `EncryptedExtensions`, `ClientFinished`, and `ServerFinished` are as for the full (EC)DHE and non–0-RTT PSK-based handshakes; we refer to Sections 6.2 and 6.4 for their description. In the following we hence focus on the new handshake messages for 0-RTT and the `draft-14` key schedule, which contains some notable changes compared to TLS 1.3 `draft-10`.

- `ClientEarlyData` (`CEAD`)/`ServerEarlyData` (`SEAD`) are extensions to the `Hello` messages sent to announce a 0-RTT handshake. The client includes the (masked) age `ticket_age` of the (ticket issuing the) used resumption secret. The server signals accepting the 0-RTT exchange with an empty `ServerEarlyData` extension.

  TLS 1.3 `draft-14` [Res16c, Section 4.2.6.2] recommends that servers should use the `ticket_age` value to check that client messages are not replayed. Depending on how well clocks are synchronized, this can prevent delayed replays, but not immediate replays. We

[31] For comparison, omitting the 0-RTT phase essentially yields the TLS 1.3 full resp. PSK-based handshake.

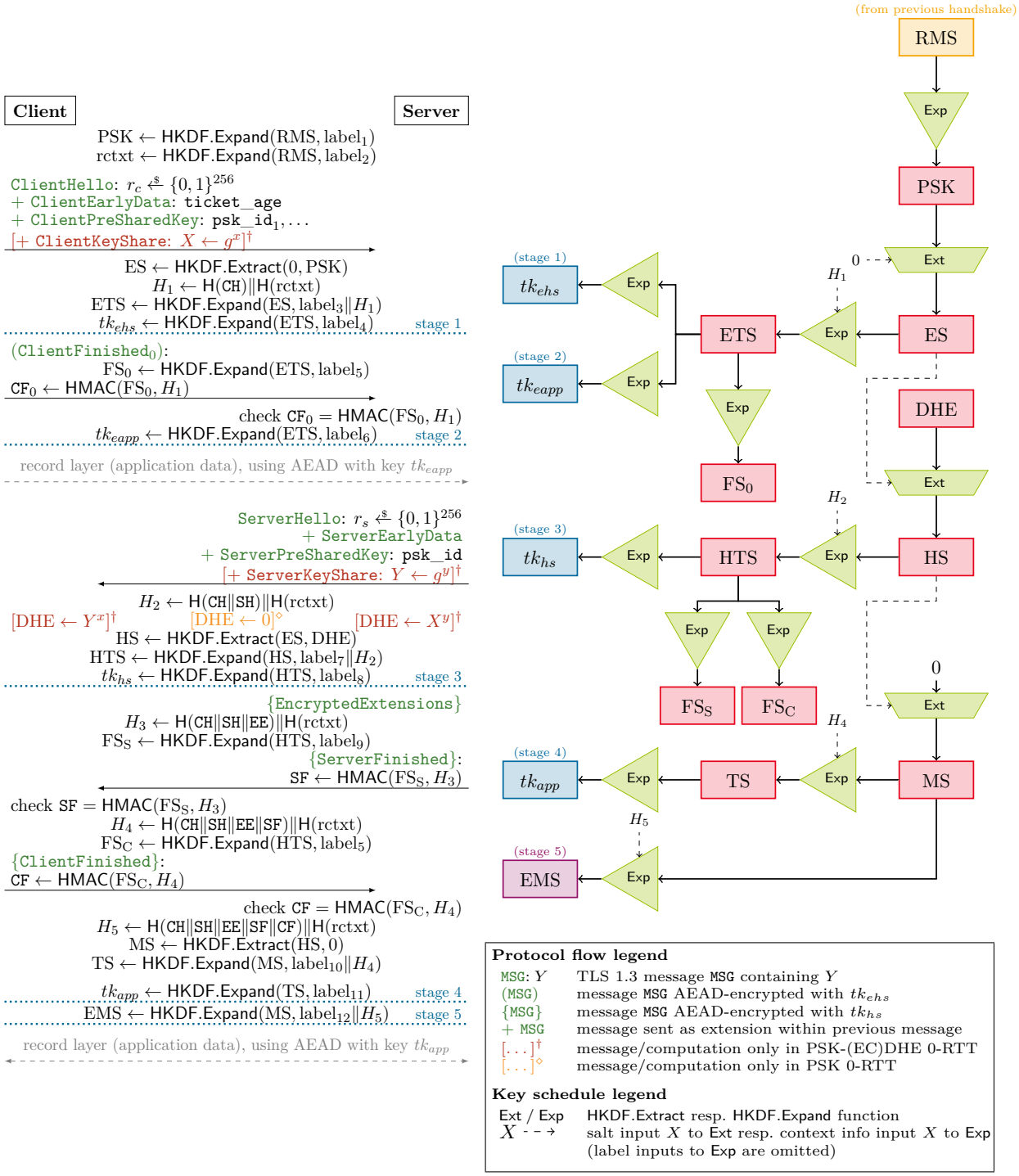


Figure 7.2: The TLS 1.3 `draft-14` PSK and PSK-(EC)DHE 0-RTT handshake protocols (left) and key schedule (right).

do not rely on this check in our analysis but conservatively treat the 0-RTT key exchange messages as arbitrarily replayable.

- For `ClientPreSharedKey`/`ServerPreSharedKey`, focusing on 0-RTT handshakes only, we only consider the case where client and server agree on the first announced `psk_id`, the one used to derive 0-RTT keys.

Based on the `ClientHello` message and extensions both sides can already derive the 0-RTT keys. First, from the shared pre-shared secret (the resumption master secret established in a previous handshake) pss = RMS a pre-shared key PSK and a resumption context value rctxt are derived using the Expand component of HKDF [Kra10, KE10]. In the key derivation, PSK serves as the starting secret while rctxt binds the derived keys to the previous handshake that established RMS. Then, the early secret ES is computed from PSK using $\mathsf{HKDF.Extract}$. Via an intermediate (expanded) early traffic secret ETS both the 0-RTT handshake and application traffic keys tk_{ehs} and tk_{eapp} are finally expanded. We again adopt here the standard notation for the two HKDF functions as introduced in Section 3.2.3.

The client then completes its first flight by sending a 0-RTT `Finished` message, sent encrypted under tk_{ehs}:

- $\texttt{ClientFinished}_0$ (CF_0) consists of an HMAC (message authentication code) value which is computed using the 0-RTT finished secret FS_0 on the (hashed) 0-RTT messages and the resumption context rctxt.

Following $\texttt{ClientFinished}_0$, the client can use tk_{eapp} to encrypt and send 0-RTT application data.[32]

After receiving the client's first flight, the server sends its `ServerHello` message along with the indicated extensions. At this point (resp. after receiving `ServerHello` for the client) both sides extract from ES the handshake secret HS (incorporating the joint Diffie–Hellman share DHE = g^{xy} in the PSK-(EC)DHE 0-RTT handshake). Again via first expanding an intermediate handshake traffic secret HTS, the handshake traffic key tk_{hs} is derived.

Server and client then complete the handshake by `EncryptedExtensions` and the `Finished` messages (keyed with distinct client resp. server finished secrets $\mathrm{FS_C}$/$\mathrm{FS_S}$ which are both expanded from HTS), encrypted under tk_{hs}.

At the end of the handshake, the master secret MS is extracted from HS and used to expand the application traffic key tk_{app} (via an intermediate traffic secret TS), used to protect the (non–0-RTT) application data sent, as well as the exporter master secret EMS which can be used to derive further key material outside of TLS.

On client authentication, 0.5-RTT data, and post-handshake messages. When analyzing the (PSK-based) 0-RTT handshake candidates for TLS 1.3, we—as in Chapter 6—focus on the main components of the handshake and hence do not capture the following more advanced options specified in `draft-14` (resp. `draft-12`, discussed below).

First, in `draft-14` the server can optionally ask the *client to authenticate* (beyond the shared secret key) by sending a public-key certificate and signing the transcript (i.e., by signature-based authentication as employed in the (EC)DHE-based handshakes of TLS 1.3). This option was removed again in later drafts and we omit it here but note that our multi-stage key exchange model can in principle be augmented to capture combined authentication under multiple long-term secrets.

[32] The server may decide to not derive any 0-RTT keys (and not accept any 0-RTT data). In that case it would, in our model, simply set the first two session identifiers sid_1, sid_2 and keys key_1, key_2 to $\perp$ and continue with deriving the third key.

Second, instead of deriving the application traffic key tk_{app} at the end of the handshake (as depicted in Figure 7.2), the server might already do so after sending the `ServerFinished` message in order to send so-called *0.5-RTT data* directly following his flight, i.e., without waiting for the `ClientFinished` response. We omit analyzing this variant of the handshake but expect that results for it with potentially weaker authentication guarantees for tk_{app} can be obtained in our model.

Third, TLS 1.3 introduces *post-handshake messages* that can be sent (potentially long) after the initial handshake was completed in order to update the used traffic key, authenticate the client, or issue tickets for session resumption. Here, we focus on the main handshake and do not consider post-handshake messages.

7.3 Security of the TLS 1.3 draft-14 PSK and PSK-(EC)DHE 0-RTT Handshakes

Our security analysis of the TLS 1.3 `draft-14` PSK and PSK-(EC)DHE 0-RTT handshakes (`draft-14-PSK-0RTT` resp. `draft-14-PSK-(EC)DHE-0RTT`) is carried out in the preshared-secret variant (MS-PSKE) of our multi-stage key exchange model. We begin with stating the protocol-specific properties $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ mostly shared by both handshakes:

- $\mathsf{M} = 5$: the PSK-based 0-RTT handshakes have five stages (deriving, in that order, traffic keys tk_{ehs}, tk_{eapp}, tk_{hs}, tk_{app}, and the exporter master secret EMS).
- the authentication properties AUTH differ between the PSK(-only) and the PSK-(EC)DHE 0-RTT handshakes:
  - for PSK 0-RTT, $\mathsf{AUTH} = \{(\mathsf{mutual}, \mathsf{mutual}, \mathsf{mutual}, \mathsf{mutual}, \mathsf{mutual})\}$: all keys established are mutually authenticated (wrt. the established pre-shared secret).
  - for PSK-(EC)DHE 0-RTT, $\mathsf{AUTH} = \{(\mathsf{mutual}, \mathsf{mutual}, \mathsf{unauth}, \mathsf{mutual}, \mathsf{mutual})\}$: the handshake traffic key tk_{hs} is unauthenticated, all other keys are mutually authenticated (wrt. the established pre-shared secret).[33]
- $\mathsf{USE} = (\mathsf{internal}, \mathsf{external}, \mathsf{internal}, \mathsf{external}, \mathsf{external})$: the (0-RTT and main) handshake traffic keys tk_{ehs} and tk_{hs} are used to protect messages within the handshake while the application traffic keys tk_{eapp} and tk_{app} as well as the exporter master secret EMS are only used externally.
- $\mathsf{REPLAY} = (\mathsf{replayable}, \mathsf{replayable}, \mathsf{nonreplayable}, \mathsf{nonreplayable}, \mathsf{nonreplayable})$: the 0-RTT stages 1 and 2 are replayable, the other stages are not.

Both TLS 1.3 `draft-14` PSK-based 0-RTT handshakes enjoy key independence for all keys. Expectedly, the PSK(-only) 0-RTT handshake provides no forward secrecy. The PSK-(EC)DHE 0-RTT handshake instead ensures forward secrecy for the non–0-RTT keys (i.e., from stage 3 on), but not for the 0-RTT keys.

[33] Although including the pre-shared secret its derivation, tk_{hs} cannot enjoy both forward secrecy and mutual authentication as the involved Diffie–Hellman shares are only authenticated after its derivation. We remark that alternatively to considering tk_{hs} being unauthenticated but forward-secret (a security property close to the notion of "weak (perfect) forward secrecy" [Kra05]), one might instead also consider tk_{hs} to be non–forward-secret but mutually authenticated.

Session matching is defined via the following session identifiers, consisting of the unencrypted messages exchanged up to each stage:

$$\mathsf{sid}_1 = (\texttt{ClientHello}),$$
$$\mathsf{sid}_2 = (\mathsf{sid}_1, \text{“EAD”}),$$
$$\mathsf{sid}_3 = (\texttt{ClientHello}, \texttt{ServerHello}),$$
$$\mathsf{sid}_4 = (\texttt{ClientHello}, \texttt{ServerHello}, \texttt{EncryptedExtensions}, \texttt{ServerFinished}), \quad \text{and}$$
$$\mathsf{sid}_5 = (\mathsf{sid}_4, \text{“EMS”}).$$

Here, `Hello` messages also comprise the sent `EarlyData`, `KeyShare`, and `PreSharedKey` extensions. We remark that, as for the analyses of the TLS 1.3 full and PSK-based handshakes (cf. Chapter 6), we too define the session identifiers over the *unencrypted* messages. This diverges from the common practice to set the session identifier as the concatenation of the (here encrypted) protocol transmissions, but is necessary to achieve key independence in the multi-stage security for such protocols.

For the contributive identifiers, we need to ensure that a server session can in any case be tested when receiving an honest client contribution (even if that client never receives the `ServerHello` response), analogously to the full handshake analysis. Hence, for stage 3, on sending resp. receiving the `ClientHello` message, client resp. server initially sets $\mathsf{cid}_3 = (\texttt{ClientHello})$ and subsequently, on receiving (resp. sending) the `ServerHello` message, extend it to $\mathsf{cid}_3 = \mathsf{sid}_3 = (\texttt{ClientHello}, \texttt{ServerHello})$. The other contributive identifiers are set to $\mathsf{cid}_i = \mathsf{sid}_i$ for $i \in \{1, 2\}$ and to $\mathsf{cid}_i = \mathsf{sid}_3$ for $i \in \{4, 5\}$ when the respective stage's session identifier is set.

We are now ready to state our security results for the PSK and PSK-DHE 0-RTT handshakes of TLS 1.3 `draft-14`. Naturally, the proof aspects concerning the non–0-RTT parts of the handshakes are structurally close to the proofs for the `draft-10` PSK-based handshakes in Section 6.5, but need to take the modified key schedule into account.

7.3.1 PSK(-only) 0-RTT Handshake

Theorem 7.1 (Match security of `draft-14-PSK-0RTT`)**.** *The TLS 1.3* `draft-14` *PSK 0-RTT handshake is* Match*-secure with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. For any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq n_s^2 \cdot 2^{-|nonce|},$$

where n_s *is the maximum number of sessions and* $|nonce| = 256$ *is the bit-length of the nonces.*

Proof. We need to prove six properties for Match security.

1. *Sessions with the same session identifier for some stage hold the same key at that stage.*
   Containing the `ClientHello` message, all session identifiers fix the used pre-shared secret identifier `psk_id` and hence the used secret $\mathsf{pss} = \mathrm{RMS}$, determining the values PSK and rctxt. Each stage's session identifier moreover fixes all messages included in the (handshake) hashes used in the key derivation of that stage's key, which is hence uniquely determined by the session identifier.

2. *Sessions with the same session identifier for some stage agree on that stage's authentication level.*
   This trivially holds as the PSK 0-RTT handshake fixes mutual authentication for all stages.

3. *Sessions with the same session identifier for some stage share the same contributive identifier.*
   For stages $i \in \{1, 2\}$ this follows immediately from $\mathsf{cid}_i = \mathsf{sid}_i$. For stages $i \in \{3, 4, 5\}$, the contributive identifier is set to its final value $\mathsf{cid}_i = \mathsf{sid}_3$ when both sides set the session identifier, which each include the messages in sid_3.

4. *Sessions are partnered with the intended (authenticated) participant and for mutual authentication share the same pre-shared secret index.*
   As honest sessions only used their own pre-shared secret identifier `psk_id`, this value included (via `ClientHello`) in all session identifiers ensures agreement of both the intended partner and key index.

5. *Session identifiers are distinct for different stages.*
   This holds trivially since session identifiers sid_1, sid_3, and sid_4 contain distinct (non-optional) messages and sid_2 and sid_5 include separating identifiers.

6. *At most two sessions have the same session identifier at any non-replayable stage.*
   We only need to consider the non-replayable stages 3–5 here.[34] The according session identifiers contain (through the `Hello` messages) randomly chosen nonces r_c and r_s (of bit-length $|\text{nonce}| = 256$) from each side, one of which a third session would need to pick by coincidence. This probability can be upper-bounded by $n_s^2 \cdot 2^{-|\text{nonce}|}$ for n_s being the maximum number of sessions. □

Theorem 7.2 (Multi-Stage security of `draft-14-PSK-0RTT`)**.** *The TLS 1.3* `draft-14` *PSK 0-RTT handshake is* Multi-Stage*-secure in a key-independent and non–forward-secret manner with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. Formally, for any efficient adversary* $\mathcal{A}$ *against the* Multi-Stage *security there exist efficient algorithms* $\mathcal{B}_1, \ldots, \mathcal{B}_9$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq 5n_s \cdot \Big(\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_p \cdot \Big(&\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_4} \\ &+ \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_7} \\ &+ \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_9}\Big)\Big),\end{aligned}$$

where n_s *is the maximum number of sessions and* n_p *is the maximum number of pre-shared secrets.*

Proof. We first restrict the adversary $\mathcal{A}$ to a single Test query. By a hybrid argument (following the detailed one for the full handshake proof from Lemma 6.3 in Section 6.3) this reduces $\mathcal{A}$'s advantage by a factor at most $1/5n_s$ for the five stages in the at most n_s sessions. It also allows to speak about *the* session label tested at stage i, which we now know in advance.

Our proof then proceeds via the following sequence of games.

Game 0. We begin with G_0 identical to the Multi-Stage game restricted to a single test query:

$$\mathsf{Adv}^{G_0}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} = \mathsf{Adv}^{\text{1-Multi-Stage}}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}}.$$

Game 1. In a first step, we exclude hash collisions by aborting whenever during the execution of honest sessions the same hash value under hash function H is computed for two distinct

[34] Observe that an adversary can indeed replay the client's first messages to multiple server sessions, resulting in the same session identifier and derived keys.

inputs. By having an algorithm $\mathcal{B}_1$ act as the challenger in Game 0 and output, when they occur, these two inputs as a collision for H, we can bound the probability of aborting by $\mathcal{B}_1$'s advantage in breaking the collision resistance of H:

$$\mathsf{Adv}^{G_0}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_1}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

Game 2. Next, we guess the (index `psk_id` of the) pre-shared secret pss employed in the tested session (i.e., the resumption master secret RMS used for this PSK handshake). Aborting on an incorrect guess, this reduces the advantage of $\mathcal{A}$ by a factor of at most the number of pre-shared secrets n_p:

$$\mathsf{Adv}^{G_1}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq n_p \cdot \mathsf{Adv}^{G_2}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}}.$$

We can now, one at a time, replace the outputs of $\mathsf{HKDF.Expand}$ and $\mathsf{HKDF.Extract}$ evaluations using RMS and derived keys by random values, leading to a sequence of according advantage bounds for their PRF security or randomness bounds of the underling HMAC function.

Game 3. We begin by replacing any $\mathsf{HKDF.Expand}$ application using $\mathsf{pss} = \mathrm{RMS}$ by evaluations of a (lazy-sampled) random function, which in particular leads to PSK and rctxt being replaced by random values $\widetilde{\mathrm{PSK}}, \widetilde{\mathrm{rctxt}} \xleftarrow{\$} \{0,1\}^\lambda$ in the tested session and any other session using the same pss.

The introduced difference in the advantage of $\mathcal{A}$ can be bounded by an adversary $\mathcal{B}_2$ against the PRF security of $\mathsf{HKDF.Expand}$ as follows. Algorithm $\mathcal{B}_2$ simulates Game 2 faithfully, but uses its PRF oracle for evaluating $\mathsf{HKDF.Expand}$ under RMS. Note that all keys derived in the PSK 0-RTT handshake are non–forward-secret and hence any (successful) adversary $\mathcal{A}$ cannot issue a $\mathsf{Corrupt}$ query on $\mathsf{pss} = \mathrm{RMS}$ used in the tested session. The employed pre-shared key is hence an unknown and uniformly random value to $\mathcal{A}$ and hence $\mathcal{B}_2$ perfectly simulates Game 2 in case its oracle computes $\mathsf{HKDF.Expand}$ and Game 3 in case its oracle is a random function.

This leads to following bound:

$$\mathsf{Adv}^{G_2}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_3}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_2}.$$

Game 4. Next, we replace values ES computed as $\mathsf{HKDF.Extract}(0, \widetilde{\mathrm{PSK}})$ by a random value $\widetilde{\mathrm{ES}} \xleftarrow{\$} \{0,1\}^\lambda$, in particular in the tested session and partnered sessions.

Recall from Section 3.2.3 that $\mathsf{HKDF.Extract}(XTS, SKM)$ is defined as $\mathsf{HMAC}(XTS, SKM)$. Assuming that for any polynomial-time algorithm $\mathcal{B}_3$ it is computationally hard to distinguish $\mathsf{HMAC}(0, SKM)$ from $X \xleftarrow{\$} \{0,1\}^\lambda$ for uniformly random chosen values $SKM \xleftarrow{\$} \{0,1\}^\lambda$, we can bound this step by $\mathcal{B}_3$'s distinguishing advantage $\mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_3}$ (cf. Definition 3.7 in Section 3.2.3). We let $\mathcal{B}_3$ simulate Game 3 as the challenger, but using its challenge value as $\mathrm{ES} = \mathsf{HKDF.Extract}(0, \widetilde{\mathrm{PSK}})$. In case this challenge is $\mathsf{HMAC}(0, SKM)$ for $SKM \in \{0,1\}^\lambda$, $\mathcal{B}_3$ simulates Game 3, if the challenge is a uniformly random value $X \xleftarrow{\$} \{0,1\}^\lambda$, $\mathcal{B}_3$ simulates Game 4.

We can hence bound this step as

$$\mathsf{Adv}^{G_3}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_4}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_3}.$$

Game 5. We next replace evaluations of $\mathsf{HKDF.Expand}$ keyed with $\widetilde{\mathrm{ES}}$ as well as $\mathsf{HKDF.Extract}$ using $\widetilde{\mathrm{ES}}$ as salt by random functions. This in particular replaces, in the tested and partnered

sessions, the early traffic and handshake secret in these sessions by random values $\widetilde{\text{ETS}}$, $\widetilde{\text{HS}} \xleftarrow{\$} \{0,1\}^\lambda$.

Observe that in both replaced Extract and Expand evaluations, $\widetilde{\text{ES}}$ is (by definition of HKDF) used to key the HMAC function, applied to a fixed label and H_1 (when expanding ETS), resp. to a fixed value 0 (when extracting HS), i.e., distinct inputs. We can hence bound the difference in the advantage of $\mathcal{A}$ introduced by this step by the advantage of an adversary $\mathcal{B}_4$ against the PRF security of HMAC, having $\mathcal{B}_4$ use its PRF oracle to compute the replaced Expand and Extract evaluations as detailed for Game 3. The resulting bound is thus:

$$\mathsf{Adv}^{G_4}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_5}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_4}.$$

Note that from now on, $\widetilde{\text{ETS}}$ is independent of any value computed in sessions that are not partnered (in stage 1 and 2) with the tested session: as such sessions do not hold the same identifiers sid_1, sid_2 and hence not the same `ClientHello` and as, by Game 1, there are no hash collisions, no non-partnered session will compute the same hash value H_1 which hence serves as a unique label in the tested and partnered sessions.

Game 6. As the next step, we replace HKDF.Expand evaluations keyed with $\widetilde{\text{ETS}}$ (in the tested and partnered session) by a lazy-sample random function, in particular replacing in those sessions the early handshake and data traffic keys as well as the client's 0-RTT finished secret by random values $\widetilde{tk_{ehs}}, \widetilde{tk_{eapp}}, \widetilde{\text{FS}_0} \xleftarrow{\$} \{0,1\}^\lambda$.

Similar to Game 3 the advantage difference induced for $\mathcal{A}$ by this step can be bound by the advantage of an adversary $\mathcal{B}_5$ against the PRF security of HKDF.Expand:

$$\mathsf{Adv}^{G_5}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_6}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5}.$$

Game 7. We again in parallel replace both HKDF.Expand and HKDF.Extract evaluations, this time keyed, resp. salted, with $\widetilde{\text{HS}}$ by random function, replacing the handshake traffic secret and master secret with random values $\widetilde{\text{HTS}}, \widetilde{\text{MS}} \xleftarrow{\$} \{0,1\}^\lambda$, in particular in the tested and partnered sessions.

As for Game 5, both evaluations are HMAC invocations keyed with $\widetilde{\text{HS}}$ and applied to distinct values (fixed label and H_2, resp. 0). We can hence likewise bound the advantage difference introduced by the PRF security of HMAC as

$$\mathsf{Adv}^{G_6}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_7}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_6}.$$

Again, as sid_3 uniquely determines the message inputs to hash value H_2 entering the derivation of HTS and, by Game 1, there are no hash collisions, $\widetilde{\text{HTS}}$ is independent of values computed in sessions not partnered with the tested session in stage 3.

Game 8. We now replace evaluations of HKDF.Expand using $\widetilde{\text{HTS}}$ (in the tested and partnered sessions) by a random function, leading to random values $\widetilde{tk_{hs}}, \widetilde{\text{FS}_\text{S}}, \widetilde{\text{FS}_\text{C}} \xleftarrow{\$} \{0,1\}^\lambda$ in those sessions. This step is again bounded by the PRF security of HKDF.Expand:

$$\mathsf{Adv}^{G_7}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_8}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_7}.$$

Game 9. Next, we replace HKDF.Expand evaluations keyed with $\widetilde{\text{MS}}$ by a random function, in particular leading to uniformly random values $\widetilde{\text{TS}}, \widetilde{\text{EMS}} \xleftarrow{\$} \{0,1\}^\lambda$ in the tested and partnered

sessions. These are moreover independent of any other values computed in sessions not partnered in stages 4 and 5, due to Game 1 and sid_4 and sid_5 fixing the inputs to H_4 and H_5. Again,

$$\mathsf{Adv}^{G_8}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_9}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8}.$$

Game 10. Finally, we replace the $\mathsf{HKDF.Expand}$ evaluations using $\widetilde{\mathrm{TS}}$ (in the tested and partnered sessions) by a random function, resulting in a random application traffic key $\widetilde{tk_{app}} \xleftarrow{\$} \{0,1\}^\lambda$, again bounded by PRF security:

$$\mathsf{Adv}^{G_9}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{10}}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_9}.$$

In Game 10, all keys derived in the tested session ($\widetilde{tk_{ehs}}$, $\widetilde{tk_{eapp}}$, $\widetilde{tk_{hs}}$, $\widetilde{tk_{app}}$, and $\widetilde{\mathrm{EMS}}$) are now chosen uniformly at random, making the Test query independent of the test bit b_{test}.

Observe furthermore that replaying a `ClientHello` message to multiple server sessions leads to all these sessions being partnered to the originating client session (and hence prevents Reveal queries). In contrast, sessions that are not partnered with the tested session (even if using the same pre-shared secret) by definition hold different session identifiers and hence use different handshake hashes (due to Game 1) as label inputs to $\mathsf{HKDF.Expand}$ in the key derivation. The resulting keys therefore are independent random values themselves, uncorrelated with the keys established in the tested session.

Therefore, $\mathcal{A}$ learns no information on the test bit b_{test} and hence

$$\mathsf{Adv}^{G_{10}}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq 0. \qquad \square$$

7.3.2 PSK-(EC)DHE 0-RTT Handshake

Theorem 7.3 (Match security of `draft-14-PSK-0RTT`)**.** *The TLS 1.3* `draft-14` *PSK-(EC)DHE 0-RTT handshake is* Match*-secure with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. For any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq n_s^2 \cdot 1/q \cdot 2^{-|nonce|},$$

where n_s *is the maximum number of sessions,* q *is the group order, and* $|nonce| = 256$ *is the bit-length of the nonces.*

Proof. For conditions 2–5 of Match security, the arguments are as for the Match security of the PSK(-only) 0-RTT handshake (cf. Theorem 7.1). We hence focus on conditions 1 and 6.

1. *Sessions with the same session identifier for some stage hold the same key at that stage.* Beyond the arguments from Theorem 7.1, the `ClientHello` and `ServerHello` messages contained in all session identifiers from stage 3 on also fix the chosen Diffie–Hellman shares g^x and g^y. This ensures agreement on HS and hence also on the keys for stages 3–5 derived (also) from these Diffie–Hellman shares.

6. *At most two sessions have the same session identifier at any non-replayable stage.* Again we can focus on the non-replayable stages 3–5 here. For the same argument as in Theorem 7.1, three (honest) sessions sharing the same session identifier requires (at least) two sessions pick the same nonce and, for PSK-(EC)DHE, also the same Diffie–Hellman share. We can upper-bound this probability by $n_s^2 \cdot 1/q \cdot 2^{-|\mathrm{nonce}|}$, where n_s is the maximum number of sessions, q is the group order, and $|\mathrm{nonce}| = 256$ is the bit-length of the nonces. $\square$

Theorem 7.4 (Multi-Stage security of `draft-14-PSK-(EC)DHE-0RTT`)**.** *The TLS 1.3* `draft-14` *PSK-(EC)DHE 0-RTT handshake is* Multi-Stage*-secure in a key-independent and stage-3-forward-secret manner with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. Formally, for any efficient adversary* $\mathcal{A}$ *against the* Multi-Stage *security there exist efficient algorithms* $\mathcal{B}_1, \ldots, \mathcal{B}_{16}$ *such that*

$$\begin{aligned}
\mathsf{Adv}^{\text{Multi-Stage},\mathcal{D}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq 5n_s \cdot \Big(& \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} \\
& + n_p \cdot \Big(\mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_2} + \mathsf{Adv}^{\text{HMAC(0,\$)-\$}}_{\mathsf{HMAC},\mathcal{B}_3} + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HMAC},\mathcal{B}_4} + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5} \Big) \\
& + n_s \cdot n_p \cdot \Big(\mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6} + \mathsf{Adv}^{\text{HMAC(0,\$)-\$}}_{\mathsf{HMAC},\mathcal{B}_7} + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HMAC},\mathcal{B}_8} \\
& \qquad + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HMAC},\mathcal{B}_9} + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{10}} + \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{HMAC},\mathcal{B}_{11}} \Big) \\
& + n_s \cdot n_p \cdot \Big(\mathsf{Adv}^{\text{snPRF-ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_{12}} + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HMAC},\mathcal{B}_{13}} + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{14}} \\
& \qquad + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{15}} + \mathsf{Adv}^{\text{PRF-sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{16}} \Big) \Big),
\end{aligned}$$

where n_s *is the maximum number of sessions and* n_p *is the maximum number of pre-shared secrets.*

Proof. Again we first restrict the adversary $\mathcal{A}$ to a single Test query (for which we now know label and stage in advance), inducing a security loss of at most $5n_s$ by a hybrid argument.

Game 1. We next exclude hash collisions by aborting the game whenever the challenger would (in honest sessions) compute the same hash value for distinct inputs. As in Game 1 for Theorem 7.2 the caused difference in the advantage can be bounded by that of an adversary $\mathcal{B}_1$ against the collision resistance of the hash function:

$$\mathsf{Adv}^{\text{1-Multi-Stage}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_1}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

Our proof then treats the following three (disjoint) cases separately:

A. the adversary tests a stage-1 or stage-2 key,

B. the adversary tests a stage-i key for $i \in \{3,4,5\}$ in a session without honest contributive partner in the third stage (i.e., there does not exist a session label' with $\mathsf{label.cid}_3 = \mathsf{label}'.\mathsf{cid}_3$ when the Test query is issued on label), and

C. the adversary tests a stage-i key for $i \in \{3,4,5\}$ in a session with honest contributive partner in the third stage.

Case A. Test in Stage 1–2

In the first proof case we are concerned with a Test query on a 0-RTT key (in stage 1 or 2). As both stages are non–forward-secret, we know that is this case no Corrupt query can have been issued for the pre-shared secret pss employed in the tested session (neither before nor after the Test query), as the adversary would otherwise lose. This allows us to apply the same proof strategy as for the Multi-Stage security of the `draft-14` PSK(-only) 0-RTT handshake in the proof Theorem 7.2. Via the very same sequence of games G_2–G_6 (starting after excluding hash collisions) we reach a game where both 0-RTT keys tk_{ehs} and tk_{eapp} are replaced by independent and uniformly random values (leaving the adversary $\mathcal{A}$ no change to win). The introduced

differences in advantage of $\mathcal{A}$ are bound as for Theorem 7.2, yielding the following overall bound for tests in stage 1 or 2 in Game 1:

$$\mathsf{Adv}^{G_1,\text{test 1–2}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq n_p \cdot \Big(\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_4} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5}\Big),$$

where $\mathcal{B}_2, \ldots, \mathcal{B}_5$ are the reduction algorithms given in the proof of Theorem 7.2.

Case B. Test in Stage 3–5 without Contributive Stage-3 Partner

We now consider the case that the Test query is issued on stage 3–5 in a (client or server) session without honest contributive partner in stage 3. Since stage 3 is unauthenticated, testing this stage would lead to immediately losing the Multi-Stage game, hence we can focus on stages 4 and 5. As we will see, given the HMAC values in the exchanged `Finished` messages are unforgeable, we can ultimately exclude that such Test queries are issued via the following sequence of games.

Game B.0. We begin with the Multi-Stage game restricted to a single Test query in stage 3–5 without contributive stage-3 partner, where collisions are excluded:

$$\mathsf{Adv}^{G_{B.0}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} = \mathsf{Adv}^{G_1,\text{stage 3–5,no } \mathsf{cid}_3\text{-partner}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}}.$$

Game B.1. We now introduce an abortion of the game as soon as a session accepts in stage 4 without honest contributive partner in stage 3. Denoting this event as $\mathsf{abort}^{G_{B.1},\mathcal{A}}_{acc}$ we can bound the induced advantage difference of $\mathcal{A}$ as

$$\left|\mathsf{Adv}^{G_{B.0}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} - \mathsf{Adv}^{G_{B.1}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}}\right| \leq \Pr[\mathsf{abort}^{G_{B.1},\mathcal{A}}_{acc}].$$

Observe that we can immediately bound $\mathsf{Adv}^{G_{B.1}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq 0$, as the game aborts before $\mathcal{A}$ has the chance to issue a Test query (recall that Test queries may only be issued to stage 4 or 5 of session without contributive stage-3 partner). We hence continue by bounding $\Pr[\mathsf{abort}^{G_{B.1},\mathcal{A}}_{acc}]$ in the remaining game sequence.

Game B.2. Next, we guess the first session that accepts in stage 4 without honest contributive stage-3 partner (i.e., the session causing $\mathsf{abort}^{G_{B.1},\mathcal{A}}_{acc}$) and abort if we guessed incorrectly. Observe that Game B.2 equals Game B.1 for a correct guess and we can hence bound

$$\Pr[\mathsf{abort}^{G_{B.1},\mathcal{A}}_{acc}] \leq n_s \cdot \Pr[\mathsf{abort}^{G_{B.2},\mathcal{A}}_{acc}],$$

where n_s is the maximum number of sessions.

Moreover, no Corrupt query can have been issued to the guessed session (or any other session using the same pre-shared secret pss): On the one hand, sessions stop execution on Corrupt and thus no such query could have been issued before the guessed session accepted in stage 4 (as otherwise it would not have accepted). On the other hand, the game aborts when the guessed session accepts in stage 4, so there is no chance for $\mathcal{A}$ to issue a Corrupt query afterwards. The pre-shared secret pss employed in the guessed session hence remains an unknown, random value for $\mathcal{A}$ which we can leverage in the following games.

Game B.3. We can now first guess the pre-shared secret $\mathsf{pss} = \mathrm{RMS}$ employed in the guessed session. Aborting on an incorrect guess introduces a factor of at most the number of pre-shared secrets n_p:

$$\Pr[\mathsf{abort}^{G_{B.2},\mathcal{A}}_{acc}] \leq n_p \cdot \Pr[\mathsf{abort}^{G_{B.3},\mathcal{A}}_{acc}].$$

Game B.4. Applying to the guessed session the steps introduced in the Games 3, 4, 5, 7, and 8 from the proof of `draft-14` PSK(-only) Multi-Stage security (Theorem 7.2), we (in particular) replace the values PSK, ES, HS, HTS, and finally $\mathrm{FS_S}$ and $\mathrm{FS_C}$ by uniformly random values sampled from $\{0,1\}^\lambda$. Denoting the resulting game by Game B.4, the introduced advantage difference is bounded as

$$\Pr[\mathsf{abort}_{acc}^{G_{B.3},\mathcal{A}}] \leq \Pr[\mathsf{abort}_{acc}^{G_{B.4},\mathcal{A}}] + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_7} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_8} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_9} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{10}},$$

where $\mathcal{B}_6, \ldots, \mathcal{B}_{10}$ are the algorithms $\mathcal{B}_2$, $\mathcal{B}_3$, $\mathcal{B}_4$, $\mathcal{B}_6$, and $\mathcal{B}_7$ given for Games 3, 4, 5, 7, and 8 in the proof of Theorem 7.2.

At this point, both server and client finished secrets used in the guessed session are replaced by uniformly random random values $\widetilde{\mathrm{FS_S}}$ resp. $\widetilde{\mathrm{FS_C}}$. As the final step, we show how this allows to turn any adversary triggering the $\mathsf{abort}_{acc}^{G_{B.4},\mathcal{A}}$ event in Game B.4 into an (EUF-CMA) MAC forger $\mathcal{B}_{11}$ for HMAC. To this extent, we let $\mathcal{B}_{11}$ act as the challenger in Game B.4, but instead of computing HMAC values using keys $\widetilde{\mathrm{FS_S}}$ or $\widetilde{\mathrm{FS_C}}$ on its own, relay them to MAC oracles of two EUF-CMA security instances for HMAC. As $\widetilde{\mathrm{FS_S}}$ and $\widetilde{\mathrm{FS_C}}$ are uniformly random values, this provides a sound simulation of Game B.4 for $\mathcal{A}$.

Recall that $\mathsf{abort}_{acc}^{G_{B.4},\mathcal{A}}$ is triggered as soon as the first (guessed) session accepts in stage 4 without contributive partner in stage 3. This in particular means that there is no session holding the same $\mathsf{cid}_3 = (\texttt{ClientHello}, \texttt{ServerHello})$ value. However, in order for the guessed session to accept in stage 4, it must have received (within `ServerFinished` or `ClientFinished`) an HMAC value on a hash covering also the messages contained in cid_3. As no session holds the same cid_3, and as there are no hash collisions (by Game 1), no honest session will have issued this HMAC value. It hence was not queried to the MAC oracle in the EUF-CMA security game (used for client resp. server messages) which allows $\mathcal{B}_{11}$ to output it as a valid forgery in the respective game. Algorithm $\mathcal{B}_{11}$ hence being successful whenever $\mathcal{A}$ triggers $\mathsf{abort}_{acc}^{G_{B.4},\mathcal{A}}$, this finally bounds

$$\Pr[\mathsf{abort}_{acc}^{G_{B.4},\mathcal{A}}] \leq \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_{11}}.$$

In summary, this provides the following advantage bounds for this proof case:

$$\mathsf{Adv}^{G_1,\text{stage 3–5,no } \mathsf{cid}_3\text{-partner}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq n_s \cdot n_p \cdot \Big(\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_7} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_8} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_9} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{10}} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_{11}}\Big).$$

Case C. Test in Stage 3–5 with Contributive Stage-3 Partner

In the last proof case we treat test sessions accepting in stages 3–5 that have an honest contributive partner in stage 3. In contrast to Case B this in particular allows the (unauthenticated) stage-3 key to be tested. Our proof strategy is geared towards leveraging the availability of honest Diffie–Hellman shares g^x and g^y (through honest cid_3-partnering) as source of randomness (unknown to $\mathcal{A}$) which ensures (forward) secrecy of the keys derived from it in stages 3–5, even if the involved pre-shared secret pss is corrupted.

Game C.0. Our initial game is the Multi-Stage game restricted to a single Test query in stage 3–5 with contributive stage-3 partner, where collisions are excluded:

$$\mathsf{Adv}^{G_{C.0}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} = \mathsf{Adv}^{G_1,\text{stage 3–5},\mathsf{cid}_3\text{-partner}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}}.$$

Game C.1. We first guess the (index of the) session contributively partnered in stage 3 with the tested session and abort on an incorrect guess, inducing a factor of n_s (the numbers of sessions):

$$\mathsf{Adv}^{G_{C.0}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq n_s \cdot \mathsf{Adv}^{G_{C.1}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}}.$$

Game C.2. Next, we guess the pre-shared secret $\mathsf{pss} = \mathrm{RMS}$ used in the tested session (and abort on an incorrect guess), reducing the advantage of $\mathcal{A}$ by a factor of n_p (the number of pre-shared secrets):

$$\mathsf{Adv}^{G_{C.1}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq n_p \cdot \mathsf{Adv}^{G_{C.2}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}}.$$

Game C.3. Knowing the (honest) session that contributes in particular the Diffie–Hellman share to stage 3 of the tested session and the used pre-shared secret in advance now allows us to encode a Diffie–Hellman challenge in the shares g^x and g^y used at the tested session.

If the tested session is a client session, we know that both it and the partnered session guessed in Game C.1 hold the same shares g^x, g^y. If the tested session however is a server session, we are not ensured that the contributive (client) partner from Game C.1 will receive the test session's share g^y unmodified.[35] It might instead receive an adversarially-controlled value $g^{y'}$ for which we then, for a correct simulation, need to be able to compute $g^{xy'}$ (while knowing neither x nor y'). To this extent, we model the security of the $\mathsf{HKDF.Extract}$ function deriving HS using ES as salt and $\mathrm{DHE} = g^{xy}$ as source key material using the $\mathsf{PRF\text{-}ODH}$ assumption [JKSS12] in its single-query $\mathsf{snPRF\text{-}ODH}$ variant. This allows us to replace HS in the tested session with a random value while still being able to compute the (potentially different) handshake secret HS' derived from $g^{xy'}$ (for an arbitrary $g^{y'} \neq g^y$).

More precisely, in Game C.3 we replace the handshake secret HS derived from ES and $\mathrm{DHE} = g^{xy}$ in the tested and (potentially) its partnered session by a uniformly random value $\widetilde{\mathrm{HS}} \xleftarrow{\$} \{0,1\}^\lambda$. The introduced advantage difference for $\mathcal{A}$ can be bound by the advantage of an algorithm $\mathcal{B}_{12}$ against the $\mathsf{snPRF\text{-}ODH}$ security of $\mathsf{HKDF.Extract}$ (using ES as salt and $\mathrm{DHE} = g^{xy}$ as source key material). To this extent, $\mathcal{B}_{12}$ outputs ES (precomputed from the test session's pre-shared secret guessed in Game C.2) as the PRF challenge label. It then employs the obtained Diffie–Hellman shares as values g^x and g^y in the `ClientKeyShare` resp. `ServerKeyShare` message of the tested session and the contributive stage-3 partner session (guessed in Game C.1). It furthermore uses the obtained PRF challenge value as the handshake secret HS in the tested session and, potentially, the partnered session (obtaining the same Diffie–Hellman shares). In case the guessed contributive partner session from Game C.1 is a client session and obtains, within `ServerKeyShare`, a value $g^{y'} \neq g^y$, $\mathcal{B}_{12}$ leverages its (single) ODH_u query in the $\mathsf{snPRF\text{-}ODH}$ game to compute HS from $g^{xy'}$ (without knowing x or y').

Recall that $\mathcal{B}_{12}$ is free to replace values g^x and g^y in the tested and contributive-partnered session at will as $\mathcal{A}$ can only passively observe them. The simulation $\mathcal{B}_{12}$ provides to $\mathcal{A}$ hence equals Game C.2 in case the PRF challenge value $\mathcal{B}_{12}$ obtains is computed as $\mathsf{HKDF.Extract}(\mathrm{ES}, g^{xy})$ while, if the challenge is a uniformly random value it equals Game C.3. Therefore,

$$\mathsf{Adv}^{G_{C.2}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.3}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_{12}}.$$

Observe that in this reduction we do not rely on (the secrecy of) the early secret ES at all. This in particular allows ES to be known to $\mathcal{A}$ through issuing a $\mathsf{Corrupt}$ query on the pre-shared secret $\mathsf{pss} = \mathrm{RMS}$ involved in the tested session at any time, thereby ensuring forward secrecy (from stage 3 on).

[35] Observe that the server's MAC value in `ServerFinished` is only processed after deriving the stage-3 key.

We complete this proof case by applying to the tested session (and its potential partner) the steps described for the Games 7, 8, 9, and 10 from the proof of `draft-14` PSK(-only) Multi-Stage security (Theorem 7.2). Thereby, we in particular replace the session keys for stages 3–5, tk_{hs}, tk_{app}, and EMS, by independent and uniformly random values sampled from $\{0,1\}^\lambda$. As the key derivation in each stage contains that stage's full session identifier, any non-partnered session moreover derives (from shared secrets) independent keys, rendering Reveal queries on such sessions useless. This leaves the adversary $\mathcal{A}$ with no chance to determine b_{test} better than through guessing and hence bounds its advantage in Game C.3 as

$$\begin{aligned}\mathsf{Adv}^{G_{C.3}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq\ & \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_{13}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{14}} \\ & + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{15}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{16}}.\end{aligned}$$

where $\mathcal{B}_{13}$, …, $\mathcal{B}_{16}$ are the algorithms $\mathcal{B}_6$, $\mathcal{B}_7$, $\mathcal{B}_8$, and $\mathcal{B}_9$ given for Games 7, 8, 9, and 10 in the proof of Theorem 7.2.

To conclude, the advantage bounds for this proof case sum up to:

$$\begin{aligned}\mathsf{Adv}^{G_1,\text{stage 3–5},\mathsf{cid}_3\text{-partner}}_{\texttt{draft-14-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq n_s \cdot n_p \cdot \Big(&\mathsf{Adv}^{\mathsf{snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_{12}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_{13}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{14}} \\ &+ \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{15}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{16}}\Big).\end{aligned}$$

□

7.4 The TLS 1.3 draft-12 (EC)DHE 0-RTT Handshake Protocol

The latest TLS 1.3 drafts do not specify a Diffie–Hellman-based ((EC)DHE) 0-RTT handshake anymore, the last draft doing so is `draft-12` [Res16a]. We nevertheless provide a security analysis of this 0-RTT mode (as specified in `draft-12`) for two reasons: For one, it is much closer to the QUIC and OPTLS protocols and our analysis hence enables a comparison with those designs. For another, it provides slightly stronger forward secrecy properties [Kra16a] as reflected in our analysis and may (for that or other reasons) be re-established as a TLS 1.3 extension [Res16e].

On a high level, the (EC)DHE 0-RTT handshake goes through the same four phases as the PSK-based 0-RTT modes: key exchange, 0-RTT, server parameters, and authentication (cf. Section 7.2). A notable difference though is that, in `draft-12`, the client may perform signature-based authentication in the 0-RTT step, an option that was abandoned in later drafts. We provide the protocol flow (and cryptographic computations) as well as the key schedule of the (EC)DHE 0-RTT handshake in Figure 7.3. To avoid repetitions, we only explain the new handshake messages for 0-RTT client authentication and the novelties in the key schedule (compared to the relatively close `draft-10` schedule), and refer to the description in Sections 6.2, 6.4, and 7.3 for the identical components in `draft-12`.

- `ClientEarlyData` (`CEAD`)/`ServerEarlyData` (`SEAD`) are extensions sent to announce a 0-RTT handshake. For `draft-12` (EC)DHE 0-RTT, the client includes an identifier `config_id` for a previously obtained server configuration (including a semi-static public key $S = g^s$ for which the server holds the secret exponent s) along with a matching ciphersuite used for deriving (and encrypting under) the 0-RTT keys.[36] The server signals accepting the 0-RTT exchange with an empty `ServerEarlyData` extension.

[36] TLS 1.3 `draft-12` again does generally not provide protection against replay of 0-RTT data between multiple connections, but allows inclusion of an optional context value within the `CEAD` message to implement unique per-connection configuration identifiers delivered out-of-band as an anti-replay measurement outside of TLS (cf. [Res16a, Section 6.3.2.5.2]). We do not consider this special mechanism in our analysis.

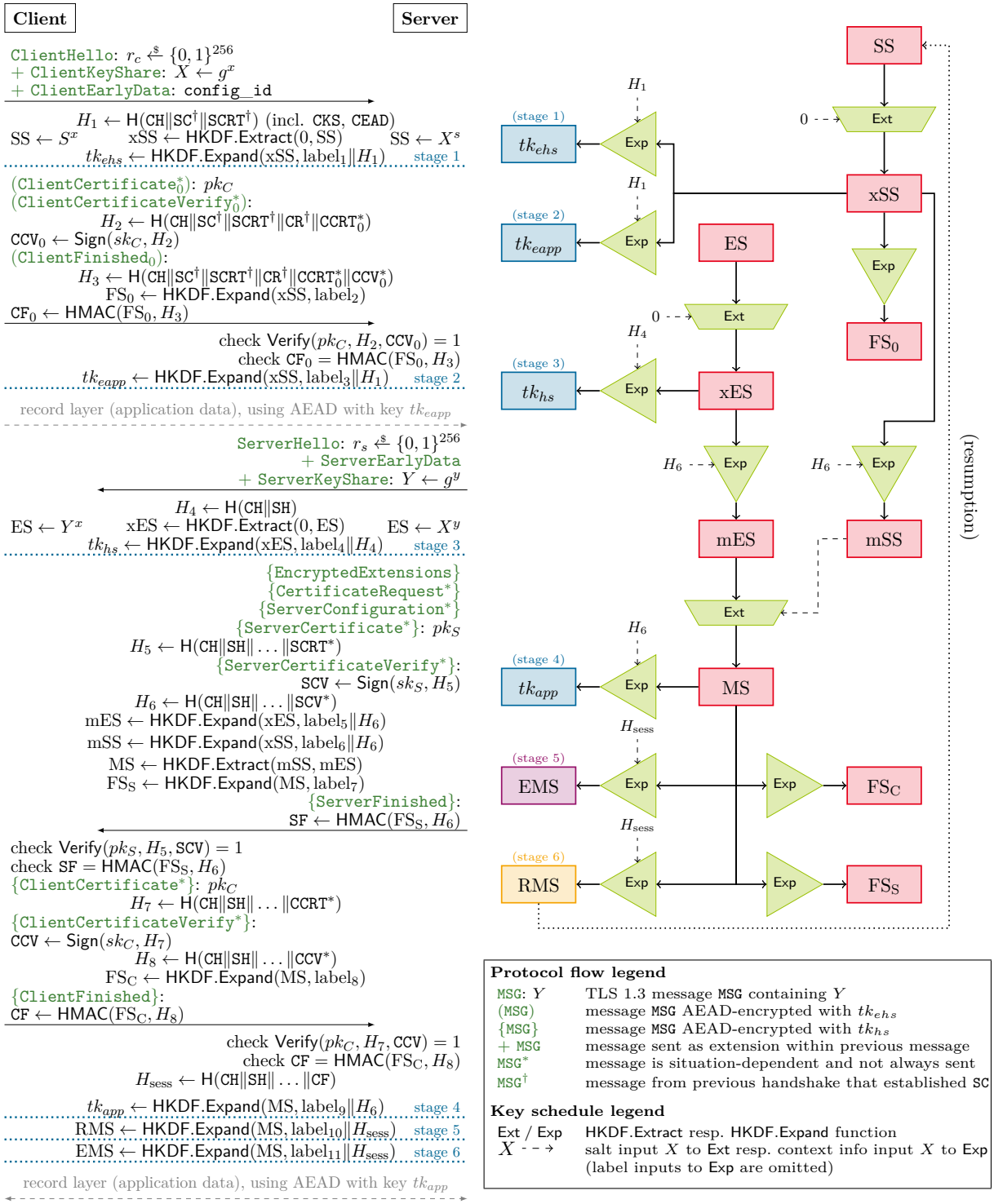


Figure 7.3: The TLS 1.3 `draft-12` (EC)DHE 0-RTT handshake protocol (left) and key schedule (right).

After sending its `ClientHello` and extensions, the client can already derive one of the two main secret inputs for key derivation, namely the *static secret* SS, as the Diffie–Hellman shared value g^{xs}. The initial 0-RTT keys are then derived using HKDF in the extract-then-expand paradigm [Kra10]: first, an extracted value xSS is computed from which, in a second step, the 0-RTT handshake and 0-RTT application data traffic keys tk_{ehs} and tk_{eapp} are expanded. We again use the common notation for the two HKDF functions as introduced in Section 3.2.3.

The following three messages of the 0-RTT phase are then sent encrypted using tk_{ehs}, which is unilaterally (server-side) authenticated. They are conceptually similar to the client's authentication messages sent later (as in the full handshake), but marked with a subscript 0 for distinction.

- $\texttt{ClientCertificate}_0$ ($\texttt{CCRT}_0$) contains the client's public-key certificate and is optionally sent if the client wishes to authenticate its 0-RTT data.

- $\texttt{ClientCertificateVerify}_0$ ($\texttt{CCV}_0$) is sent only if $\texttt{ClientCertificate}_0$ is sent and contains a digital signature over the *handshake hash* (the hash of all sent and received handshake messages at this point, i.e., here only the client's messages `CH` (incl. extensions) and $\texttt{CCRT}_0$), as well as the server configuration used by the client, and further messages from the previous handshake in which the client received this server configuration.

- $\texttt{ClientFinished}_0$ ($\texttt{CF}_0$) contains an HMAC value, a message authentication code computed over (a hash of) the same messages as for $\texttt{CCV}_0$ and the $\texttt{CCV}_0$ message itself; keyed with the 0-RTT finished secret FS_0 derived from xSS.

At this point, the client can send early (0-RTT) application data, encrypted under tk_{eapp} (which is server- and optionally also client-authenticated).

After receiving the client's first flight, the server can derive the same 0-RTT keys using its stored configuration for the semi-static key g^s and process the client's 0-RTT handshake and application data. The server can also decide to reject 0-RTT keys and data (setting sid_1, sid_2, key_1, and key_2 to $\perp$ in our model); it must in particular do so if the server configuration identifier `config_id` sent by the client is invalid, unknown, or expired. In either case, the server continues the handshake by sending out its `ServerHello` message and extensions.

Both parties can then compute the second secret input, the *ephemeral secret* ES, as the shared Diffie–Hellman value g^{xy}, and an extracted value xES from which the (unauthenticated) handshake traffic key tk_{hs} is expanded. The key tk_{hs} is then used to encrypt the remaining handshake, which is as for in the `draft-10` full (EC)DHE handshake (see Section 6.2).

After the `ServerCertificateVerify` message is sent resp. received, both parties derive the master secret MS, extracted from the intermediate expanded versions mES and mSS of the (extracted) ephemeral and static secrets xES and xSS). At the end of the handshake, three final keys are derived from the master secret through HKDF expansion steps: the application traffic key tk_{app} for protecting the (non–0-RTT) application data sent[37], the resumption master secret RMS for later pre-shared key–based session resumption, and the exporter master secret EMS from which further key material for usage outside of TLS can be derived.

As for the PSK-based 0-RTT handshakes, our analysis of the `draft-12` (EC)DHE 0-RTT handshake focuses on its core components and we hence do not treat 0.5-RTT data and post-handshake messages.

[37]More precisely, TLS 1.3 derives an intermediate *traffic secret* from which the actual application traffic key tk_{app} is expanded. This is done in order to allow for later key updates, where first an updated traffic secret is computed from which then the new traffic key is derived. We do not capture this key update mechanism and hence omit the intermediate traffic secret derivation here for simplicity.

7.5 Security of the TLS 1.3 draft-12 (EC)DHE 0-RTT Handshake

We conduct our security analysis of the TLS 1.3 `draft-12` (EC)DHE 0-RTT handshake (`draft-12-(EC)DHE-0RTT`) in the public-key variant (MSKE) of the multi-stage key exchange model. First, we need to state the (formalized) protocol-specific properties (such as the available authentication modes, its replayability properties, etc.) as well as how session matching is defined via the session and contributive identifiers for each stage. The properties are captured as follows:

- $\mathsf{M} = 6$: the (EC)DHE 0-RTT handshake has six stages (deriving, in that order, traffic keys tk_{ehs}, tk_{eapp}, tk_{hs}, tk_{app}, as well as resumption and exporter master secrets RMS and EMS).

- $\mathsf{AUTH} = \{(\mathsf{unilateral}, \mathsf{auth}_{\mathrm{eapp}}, \mathsf{unauth}, \mathsf{auth}_{\mathrm{fin}}, \mathsf{auth}_{\mathrm{fin}}, \mathsf{auth}_{\mathrm{fin}}) \mid \mathsf{auth}_{\mathrm{eapp}} \in \{\mathsf{unilateral}, \mathsf{mutual}\}, \mathsf{auth}_{\mathrm{fin}} \in \{\mathsf{unauth}, \mathsf{unilateral}, \mathsf{mutual}\}\}$: the first-stage key is always unilaterally authenticated, the second-stage (early-data) key can bei unilaterally or mutually authenticated, the traffic handshake key is always unauthenticated, and the final three keys (tk_{app}, RMS, and EMS) share the same authentication property which can be no authentication, unilateral authentication, or mutual authentication.

- $\mathsf{USE} = (\mathsf{internal}, \mathsf{external}, \mathsf{internal}, \mathsf{external}, \mathsf{external}, \mathsf{external})$: the early-data and regular handshake traffic keys tk_{ehs} and tk_{hs} are used within the handshake, whereas the early-data and main application traffic keys tk_{eapp} and tk_{app} as well as the resumption and exporter master secret RMS and EMS are used only externally.

- $\mathsf{REPLAY} = (\mathsf{replayable}, \mathsf{replayable}, \mathsf{nonreplayable}, \mathsf{nonreplayable}, \mathsf{nonreplayable}, \mathsf{nonreplayable})$: the early-data stages 1 and 2 are replayable, the other stages are not.

As we will see, the TLS 1.3 `draft-12` 0-RTT handshake furthermore enjoys key independence and forward secrecy (wrt. compromise of the long-term signing secrets) for all keys. For the forward secrecy of the early-data (0-RTT) keys tk_{ehs} and tk_{eapp}, recall that our model treats compromises of long-term and semi-static secrets independently through the Corrupt resp. RevealSemiStaticKey query. While those keys remain (forward) secret after a long-term key compromise, they are replayable and hence become insecure when the involved semi-static key is revealed.[38]

For session matching, we define the session identifiers for the first four stages to be essentially the unencrypted messages sent and received up to (excluding) `Finished` that enter the handshake hash for the respective key's derivation, including (for the early-data stages) the `ServerConfiguration` and accompanying messages the client received earlier. Notably, we treat the stage-4 application traffic key tk_{app} as mutually authenticated despite being derived from a handshake hash only up to `ServerFinished` (to enable 0.5-RTT communication), and hence include in sid_4 the client's certificate and signature. As we will see in the proof of Multi-Stage security, the key derivation of tk_{app} not including the client's authentication message will induce an additional EUF-CMA security bound for HMAC compared to the proof for `draft-10` (we remark that this addition corrects the original proof in [FG17]).

[38] For this reason, our result in particular does not contradict the statement in `draft-12` that "[t]his [0-RTT] data is not forward secret, because it is encrypted solely with the server's semi-static (EC)DH share" [Res16a, Section 6.2.2]. The draft merely requires a forward-secret key to be resilient against compromises of both long-term *and* semi-static keys.

The session identifiers are hence defined as follows:

$$\begin{aligned}
\mathsf{sid}_1 &= (\texttt{ClientHello}, \texttt{ServerConfiguration}^{\dagger}, \texttt{ServerCertificate}^{\dagger}),\\
\mathsf{sid}_2 &= (\texttt{ClientHello}, \texttt{ServerConfiguration}^{\dagger}, \texttt{ServerCertificate}^{\dagger}, \texttt{CertificateRequest}^{\dagger},\\
&\quad\ \texttt{ClientCertificate}_0^{*}, \texttt{ClientCertificateVerify}_0^{*}),\\
\mathsf{sid}_3 &= (\texttt{ClientHello}, \texttt{ServerHello}), \quad \text{and}\\
\mathsf{sid}_4 &= (\texttt{ClientHello}, \texttt{ServerHello}, \texttt{EncryptedExtensions}, \texttt{CertificateRequest}^{*},\\
&\quad\ \texttt{ServerConfiguration}^{*}, \texttt{ServerCertificate}^{*}, \texttt{ServerCertificateVerify}^{*},\\
&\quad\ \texttt{ServerFinished}, \texttt{ClientCertificate}^{*}, \texttt{ClientCertificateVerify}^{*}).
\end{aligned}$$

Here, `Hello` messages always contain the according `KeyShare` and `EarlyData` extensions, components marked with † are those enabling the 0-RTT exchange received by the client in an earlier handshake, and starred (*) components are not present in all authentication modes. For the two final stages, we define the session identifiers to contain a distinguishing label beyond the full stage-4 identifier, namely $\mathsf{sid}_5 = (\mathsf{sid}_4, \text{“RMS”})$ and $\mathsf{sid}_6 = (\mathsf{sid}_4, \text{“EMS”})$.

As for the PSK-based handshakes (cf. Section 7.3), we define the contributive identifiers such that a server session can be tested when receiving an honest client contribution. That is, for stage 3 (deriving the handshake traffic key), we let the client (resp. server) on sending (resp. receiving) the `ClientHello` message (including the `ClientKeyShare` and `ClientEarlyData` extension) initially set $\mathsf{cid}_3 = (\texttt{ClientHello})$ and subsequently, on receiving (resp. sending) the `ServerHello` message (incl. `SKS`, `SEAD`), extend it to $\mathsf{cid}_3 = \mathsf{sid}_3 = (\texttt{ClientHello}, \texttt{ServerHello})$. The other contributive identifiers are set to $\mathsf{cid}_i = \mathsf{sid}_i$, for stages $i \in \{1, 2\}$ when sending resp. receiving the 0-RTT `ClientFinished` message and to $\mathsf{cid}_i = \mathsf{sid}_3$ for stages $i \in \{4, 5, 6\}$ by each party on sending its respective `Finished` message.

Finally, we capture as semi-static public and private keys the values g^s, resp. s, incorporated in the derivation of the static secret SS and 0-RTT key derivation, issued by servers and learned by clients via some `ServerConfiguration` (`SC`) message in an earlier (EC)DHE full or 0-RTT handshake. This message is sent together with a `ServerCertificate` (`SCRT`) and, optionally, `CertificateRequest` (`CR`) message (the latter enabling 0-RTT client authentication) and is signed within the `ServerCertificateVerify` message. In our model, we let the adversary control the generation of semi-static keys through the NewSemiStaticKey query, deciding whether a `CR` message is sent or not by setting the optional input $\mathsf{sskaux}_{\mathsf{pre}} = \texttt{CR}$. The NewSemiStaticKey query then samples an exponent value s at random to generate a new semi-static key, and outputs the auxiliary information $\mathsf{sskaux} = (\texttt{SC}, \texttt{SCRT})$, resp. $\mathsf{sskaux} = (\texttt{SC}, \texttt{SCRT}, \texttt{CR})$, along with $\mathsf{sspk} = g^s$ and a key identifier sskid. When instantiating a new session (through NewSession), the adversary controls which semi-static key a client session uses for the 0-RTT exchange via the sskid_V identifier and determines whether a server issues a new semi-static key in a `ServerConfiguration` message (and which key this is) via the server sessions' sskid_U identifier.[39] Finally, the RevealSemiStaticKey query allows the adversary to learn the secret exponent s for semi-static keys of its choice, at the price of not being allowed to test stage-1 and stage-2 (i.e., early-data) keys anymore for which this semi-static key was used.

We are now able to provide our security results for the TLS 1.3 `draft-12` (EC)DHE 0-RTT handshake.

Theorem 7.5 (Match security of `draft-10-(EC)DHE`). *The TLS 1.3* `draft-12` *(EC)DHE 0-RTT handshake is* Match*-secure with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. For any*

[39] Note that in TLS 1.3 only servers hold semi-static keys. In particular, NewSession queries in our model will hence have the client's semi-static key identifier (sskid) parameter set to $\perp$.

efficient adversary $\mathcal{A}$ we have

$$\mathsf{Adv}^{\mathsf{Match}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq n_s^2 \cdot 1/q \cdot 2^{-|nonce|},$$

where n_s is the maximum number of sessions, q is the group order, and $|nonce| = 256$ is the bit-length of the nonces.

Proof. We need to show that the six conditions for Match security hold:

1. *Sessions with the same session identifier for some stage hold the same key at that stage.* The session identifiers for stages 1–2 and stages 3–6 contain the client's resp. the client's and the server's `Hello` messages and hence fix the Diffie–Hellman shares g^x and g^s (through the unique semi-static key/con-figuration identifier sskid = `config_id`) resp. also g^y. Therefore, coinciding session identifiers imply agreement on the static secret SS (in stages 1–2) and, for stage 3–4, also the ephemeral secret ES (i.e., the input keying material). As each session identifier in particular furthermore fixes all messages exchanged that enter the handshake hash within the key derivations, the session keys are uniquely determined by the session identifier in each stage.

2. *Sessions with the same session identifier for some stage agree on that stage's authentication level.*
   For stages 1 and 3, the authentication level is fixed to unilateral resp. unauth. For the other stages, unilateral authentication is indicated (exactly) by the exchange of `SCRT` and `SCV` messages, whereas mutual authentication requires messages `CCRT` and `CCV` (as well as, for stages 4–6, also the server's messages). Hence, agreement on the session identifier (including theses messages) implies agreement on that stage's authentication.

3. *Sessions with the same session identifier for some stage share the same contributive identifier.*
   For stages $i \in \{1, 2\}$ this follows immediately from $\mathsf{sid}_i = \mathsf{cid}_i$. For stages $i \in \{3, 4, 5, 6\}$, note that when both sides set the session identifier, the contributive identifier is also set to its final value $\mathsf{cid}_i = \mathsf{sid}_3$, with the messages in sid_3 being contained in sid_i.

4. *Sessions are partnered with the intended (authenticated) participant.*
   As sessions of honest parties will not attest a different identity than their own in the `SCRT` and `CCRT` messages nor accept such a message for an identity different from the intended partner, agreement on these messages (which are included in the respective session identifiers for unilaterally and mutually authenticated stages) in particular implies agreement on each partner's identity.

5. *Session identifiers are distinct for different stages.*
   This holds trivially since session identifiers sid_1–sid_4 contain distinct (non-optional) messages and sid_5 and sid_6 include a separating identifier.

6. *At most two sessions have the same session identifier at any non-replayable stage.*
   As stages 1 and 2 are replayable, we need to consider this condition only for the stages 3–6.[40] Observe that the session identifiers for those stages contain the client's and server's `Hello` message and, hence, for each side a randomly chosen nonce (r_c, resp. r_s) as well as

[40] Note that, indeed, a server has no means to check whether the client's first-flight messages (being the only content of the first two stages' session identifiers) have been transmitted to another server before. This allows an adversary to replay those messages and, hence, make one client session be partnered with multiple server sessions, all deriving the same session key. As we will see in a moment, this still does not allow the adversary to break those keys' secrecy.

a randomly chosen group element (g^x, resp. g^y). Therefore, in order for a third session to agree on the same session identifier it needs to, at least, pick the same nonce and group element as the client or server, which can be upper bounded by the probability of any two parties colliding on the same nonce and group element. This probability can be bounded from above by $n_s^2 \cdot 1/q \cdot 2^{-|\text{nonce}|}$, where n_s is the number of all (client or server) sessions, q is the group order, and $|\text{nonce}| = 256$ the bit-length of the nonces. □

Theorem 7.6 (Multi-Stage security of `draft-12-(EC)DHE-0RTT`)**.** *The TLS 1.3* `draft-12` *(EC)DHE 0-RTT handshake is* Multi-Stage*-secure in a key-independent and stage-1-forward-secret manner with properties* (M, AUTH, USE, REPLAY) *given above. Formally, for any efficient adversary* $\mathcal{A}$ *against the* Multi-Stage *security there exist efficient algorithms* $\mathcal{B}_1, \ldots, \mathcal{B}_{11}$ *such that*

$$\begin{aligned}
\mathsf{Adv}^{\text{Multi-Stage},\mathcal{D}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq 6n_s \cdot \Big(& \mathsf{Adv}^{\text{COLL}}_{\text{H},\mathcal{B}_1} \\
& + n_u \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\text{Sig},\mathcal{B}_2} \\
& + n_s \cdot n_{ss} \cdot \Big(\mathsf{Adv}^{\text{msPRF-ODH}}_{\text{HKDF.Extract},\mathbb{G},\mathcal{B}_3} + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_4} + \mathsf{Adv}^{\text{EUF-CMA}}_{\text{HMAC},\mathcal{B}_5} \Big) \\
& + n_u \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\text{Sig},\mathcal{B}_6} \\
& + n_s \cdot \Big(\mathsf{Adv}^{\text{snPRF-ODH}}_{\text{HKDF.Extract},\mathbb{G},\mathcal{B}_7} + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_8} + \mathsf{Adv}^{\text{st-Extract}}_{\text{HKDF.Extract},\mathcal{B}_9} \\
& \qquad + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_{10}} + \mathsf{Adv}^{\text{EUF-CMA}}_{\text{HMAC},\mathcal{B}_{11}} \Big) \Big),
\end{aligned}$$

where n_u *is the maximum number of users,* n_s *is the maximum number of sessions, and* n_{ss} *is the maximum number of semi-static keys.*

Proof. First of all we consider that the adversary $\mathcal{A}$ makes a single Test query only. This reduces its advantage, based on a hybrid argument, by a factor at most $1/6n_s$ for the six stages in each of the n_s sessions. We can now speak about *the* session label tested at stage i, and that we know the index of the session and the stage in advance.

Game 1. We first, as in the proofs before, exclude hash collisions by aborting the game whenever the challenger in the game with a single test query would (in honest sessions) compute the same hash value for distinct inputs. The caused difference in the advantage can be bounded by that of an adversary $\mathcal{B}_1$ against the collision resistance of the hash function:

$$\mathsf{Adv}^{\text{1-Multi-Stage}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_1}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} + \mathsf{Adv}^{\text{COLL}}_{\text{H},\mathcal{B}_1}.$$

In our analysis, we then separately treat the (disjoint) cases that the adversary tests an early-data (stage-1 or stage-2) key or a regular key (stages 3–6). For tests on stages 1 and 2, we further distinguish between the following two (again disjoint) sub-cases that

A. the adversary tests a server session without honest (contributive) partner in the first stage (i.e., $\mathsf{label.role} = \mathsf{responder}$ for the test session label and there exists no $\mathsf{label}' \neq \mathsf{label}$ with $\mathsf{label.cid}_1 = \mathsf{label}'.\mathsf{cid}_1$) and

B. the adversary tests a server session with honest (contributive) partner in the first stage or a client session (i.e., $\mathsf{label.role} = \mathsf{initiator}$ or $\mathsf{label.role} = \mathsf{responder}$ and there exists a $\mathsf{label}' \neq \mathsf{label}$ with $\mathsf{label.cid}_1 = \mathsf{label}'.\mathsf{cid}_1$).

For tests on stages 3–6, we split our analysis along the same two (sub)cases used in the analysis of the full (EC)DHE handshake for `draft-10` (cf. Theorem 6.2 in Section 6.3). As we detail in

the proof, the analysis mostly follows closely the one for draft-10, so that we obtain similar security bounds, except for an added EUF-CMA security bound for HMAC due to the modified 0.5-RTT key derivation of tk_{app} (we remark that this added bound was missing in the original proof in [FG17] which we correct here).

C. the adversary tests a (client or server) session without honest contributive partner in the third stage (i.e., for the test session label there exists no $\mathsf{label}' \neq \mathsf{label}$ with $\mathsf{label.cid}_3 = \mathsf{label}'.\mathsf{cid}_3$), and

D. the tested session has an honest contributive partner in stage 3 (i.e., there exists label' with $\mathsf{label.cid}_3 = \mathsf{label}'.\mathsf{cid}_3$).

This allows us to split the adversary's advantage along these four cases:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq 6n_s \cdot \Big(&\mathsf{Adv}^{G_1,\text{test 1–2,server without partner}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \\ &+ \mathsf{Adv}^{G_1,\text{test 1–2,server with partner/client}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \\ &+ \mathsf{Adv}^{G_1,\text{test 3–6,test without partner}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \\ &+ \mathsf{Adv}^{G_1,\text{test 3–6,test with partner}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}}\Big).\end{aligned}$$

For the proof of each case we will proceed in a sequence of games. We start from the original Multi-Stage game with a single Test query and excluded hash collision (i.e., Game 1), restricted to the case in question, and modify this game in each step, showing that the difference in the adversary's advantage between the two games can be bounded by complexity-theoretic assumptions. Finally, in the last game the advantage of $\mathcal{A}$ will be at most 0 and the advantage for the considered case hence bound by combination of the intermediate bounds.

Case A. Stage 1–2: Test Server without Stage-1 Partner

For the case that the adversary tests a server (responder) session in stage 1 or 2 without contributive partner in the first stage, we recall that $\mathsf{cid}_i = \mathsf{sid}_i$ for $i \in \{1, 2\}$ and hence, as all messages in sid_1 are also contained in sid_2, the tested session also cannot have a (contributive) partner in the second stage. In order to not lose immediately, the adversary can test responder session stages without contributive partner only if they are mutually authenticated. Since the stage-1 keys are unilaterally authenticated we can focus on the second stage to be tested and assume $\mathsf{label.auth}_2 = \mathsf{mutual}$.

Game A.0. We begin with the initial game $G_{A.0}$ which equals the Multi-Stage game with excluded hash collisions and one Test query which must be issued on a stage-1 or stage-2 key of a server session without honest contributive identifier. Therefore,

$$\mathsf{Adv}^{G_{A.0}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} = \mathsf{Adv}^{G_1,\text{test 1–2,server without partner}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}}.$$

Game A.1. Our only change now is to let the challenger abort the game if the tested server session receives a $\texttt{ClientCertificateVerify}_0$ message which contains a valid signature (under some public key pk_U) over the expected (hashed) messages $H_2 = \mathsf{H}(\texttt{CH}\|\texttt{SC}^\dagger\|\texttt{SCRT}^\dagger\|\texttt{CR}^\dagger\|\texttt{CCRT}^*_0)$ for which no honest (client) session ever computed a signature.

We can bound the probability of such an event by the advantage of an adversary $\mathcal{B}_2$ against the unforgeability (in the sense of EUF-CMA) of the signature scheme Sig. In the reduction,

$\mathcal{B}_2$ first needs to guess the identity $U \in \mathcal{U}$ under whose public key the obtained signature will verify, replacing that user's public key with the challenge public key in the EUF-CMA game and generating signatures using the signing oracle. All other keys are generated by $\mathcal{B}_2$ during the game setup. When the tested session receives a valid $\texttt{CCV}_0$ message causing an abort, $\mathcal{B}_2$ outputs the contained signature as forgery.

As the tested session has no honest partner, no honest client signed the contained handshake hash before. Otherwise, this client would agree on all messages up to $\texttt{CCRT}_0$ and hence also on the first-stage session identifier (which we excluded in this proof case). Moreover, no party with a different session identifier signed the same handshake hash, as this would constitute a hash collision which we excluded in Game 1. Given that $\mathcal{B}_2$ correctly guessed the used public key pk_U (among the keys of the at most n_u users), its output constitutes a valid forgery and hence

$$\mathsf{Adv}^{G_{A.0}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{A.1}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2}.$$

At this point, we are ensured that an honest client issued a $\texttt{ClientCertificateVerify}_0$ signature on $H_2 = \mathsf{H}(\texttt{CH}\|\texttt{SC}^\dagger\|\texttt{SCRT}^\dagger\|\texttt{CR}^\dagger\|\texttt{CCRT}_0^*)$ hold by the tested session. As there are no hash collisions, this client hence in particular agrees on all messages included in the stage-1 contributive identifier $\mathsf{cid}_1 = \mathsf{sid}_1 = (\texttt{CH}, \texttt{SC}^\dagger, \texttt{SCRT}^\dagger)$. Therefore, the tested (server) session cannot be without (contributive) partner in stage 1 and hence $\mathcal{A}$ cannot issue a Test query anymore at this point, leaving the test bit unknown to $\mathcal{A}$ and thus

$$\mathsf{Adv}^{G_{A.1}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq 0.$$

Case B. Stage 1–2: Test Server with Stage-1 Partner or Client

In the second case for tests on early-data keys, we know that a tested server session always has a partnered session and that, for client sessions, the server is always authenticated (through the server configuration of some previous communication). A Test query can in this case be issued in any of the two stages.

Game B.0. We start with the unmodified initial game $G_{B.0}$:

$$\mathsf{Adv}^{G_{B.0}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} = \mathsf{Adv}^{G_1,\text{test 1–2,server with partner/client}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}}.$$

Game B.1. First, we guess the (index of the) client session involved in the test (i.e., the client session itself if the client-side is tested, or the client session partnered with the server session if the server side is tested) and abort if this guess is incorrect. Note that in both cases, this client session is an honest session simulated by the challenger. This can reduce the adversary's advantage by a factor of at most the number of sessions n_s:

$$\mathsf{Adv}^{G_{B.0}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq n_s \cdot \mathsf{Adv}^{G_{B.1}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}}.$$

Game B.2. Next, we additionally guess the (index of the) configuration identifier `config_id` (i.e., in terms of our model, the semi-static key identifier sskid resp. the NewSemiStaticKey query through which it is issued) the involved client session will use within its `ClientHello` message. Again, this reduces the advantage of $\mathcal{A}$ by a factor of at most the number of semi-static keys n_{ss}:

$$\mathsf{Adv}^{G_{B.1}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq n_{ss} \cdot \mathsf{Adv}^{G_{B.2}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}}.$$

Game B.3. At this point, we know in advance which semi-static key the involved client session will employ, enabling us to encode a Diffie–Hellman challenge in the static secret SS derived from the client's ephemeral share g^x and the server's semi-static share g^s (selected through the server configuration identifier).

As the server's semi-static key g^s is potentially used in more than one session, we need to be able to compute (keys from) further static secrets SS′ from the same g^s but a different $g^{x'}$, even when knowing neither s nor x'. Moreover, when the client session continues, it might obtain, within the `ServerKeyShare` message, an ephemeral Diffie–Hellman share $g^{y'}$ different from the value g^y chosen by the honest server session partnered in the early-data stages.[41] We hence, once, need to compute (keys from) the ephemeral secret ES (resp. xES) in the client session, even without knowing x or y'. To this end, we model the HKDF.Extract function as a pseudorandom function keyed (as source key material) with elements from group $\mathbb{G}$ and employ the PRF-ODH assumption in its msPRF-ODH variant (cf. Definition 3.6 in Section 3.2.2). The latter allows us to replace a HKDF/PRF value under SS $= g^{xs}$ with a random value while providing oracle access to evaluate the PRF under further keys SS′ $= g^{x's}$ (for $g^{x'} \neq g^x$) as well as, once, under a key ES′ $= g^{xy'}$ (for an arbitrary $g^{y'} \neq g^s$, without knowing x).

More in detail, in Game B.3 we replace the extracted static secret xSS by a uniformly random string $\widetilde{\text{xSS}} \xleftarrow{\$} \{0,1\}^\lambda$ in both the tested and its potentially partnered session(s), as well as in any session that sends or receives within the `ClientHello` message the same ephemeral key g^x and (identifier for the) semi-static key g^s (and ignore the SS value in those sessions). We can bound the difference in advantage of adversary $\mathcal{A}$ through this modification by the advantage of an algorithm $\mathcal{B}_3$ in winning the msPRF-ODH game as follows.

Initially, $\mathcal{B}_3$ receives a group element g^u in the msPRF-ODH game and immediately issues the challenge query $x^\star = 0$ for which it obtains a response $(g^v, y^\star)$ where $y^\star$ is either the value $\mathsf{PRF}(g^{uv}, 0)$ or a uniformly random string. It then acts as the challenger in the Multi-Stage game for $\mathcal{A}$, choosing a test bit $b_{\mathsf{test}} \xleftarrow{\$} \{0,1\}$ at random and simulating it according to the description except for the following changes.

- When $\mathcal{A}$ issues the NewSemiStaticKey query guessed in Game B.2, algorithm $\mathcal{B}_3$ uses g^u for the returned semi-static public key sspk (implicitly setting $g^s = g^u$ for the tested session).

- For the tested session and its potential partnered session(s), $\mathcal{B}_3$ uses g^v as the ephemeral Diffie–Hellman share of the guessed tested client session resp. partnered client session of the tested server session, implicitly setting $g^x = g^v$ for the tested session. Furthermore, we use xSS $= y^\star$ as the extracted semi-static secret in both the tested and potential partnered session(s) as well as in server sessions obtaining the same ephemeral g^x and previous `ServerConfiguration` for g^s.

- For any server session using the guessed semi-static key that obtains a client ephemeral Diffie–Hellman share $g^{x'} \neq g^x$, algorithm $\mathcal{B}_3$ does not compute SS explicitly but uses xSS $\leftarrow \mathsf{PRF}((g^{x'})^u, 0)$ directly as the response of a query $\mathsf{ODH}_u(g^{x'}, 0)$.

- For the client session tested or partnered with the tested server session in stage 1, algorithm $\mathcal{B}_3$ does not explicitly compute the ephemeral secret ES. Instead, it directly uses the response $\mathsf{PRF}((g^{y'})^v, 0) = \mathsf{PRF}((g^{y'})^x, 0)$ to one single query $\mathsf{ODH}_v(g^{y'}, 0)$ as xES $\leftarrow \mathsf{PRF}((g^{y'})^x, 0)$, where $g^{y'}$ is the server's key share obtained by the client within the `ServerKeyShare` message.

[41] Observe that, while the server might sign its share, this signature is only checked *after* the (always unauthenticated) handshake traffic key is computed and, hence, might contain an adversarially controlled server share $g^{y'}$.

In the special case that $g^{y'} = g^s$ (resulting in SS = ES for the client session), algorithm $\mathcal{B}_3$ however does not issue a query, but simply (re-)uses the challenge xES = xSS = $y^\star$.

Finally, $\mathcal{B}_3$ outputs 1 if $\mathcal{A}$ wins in the Multi-Stage game and otherwise 0.

In case $y^\star = \mathsf{PRF}(g^{uv}, 0)$ this approach equals Game B.2 while if $y^\star$ is a uniformly random value, it equals Game B.3. For this, let us see why $\mathcal{B}_3$ provides correct simulations for $\mathcal{A}$ in both cases.

First of all, recall that implicitly $g^s = g^u$ and $g^x = g^v$ in the tested and partnered sessions, hence xSS $\leftarrow \mathsf{PRF}(g^{uv}, 0) = \mathsf{PRF}(g^{xs}, 0)$ is chosen as in the real Game B.2 in case $b = 0$ in the msPRF-ODH game (recall that HKDF.Extract is our pseudorandom function). In case $b = 1$, the value xSS $\xleftarrow{\$} \{0,1\}^\lambda$ is a randomly chosen element as specified for Game B.3.

Moreover, $\mathcal{B}_3$ is free to replace these two values at will, as $\mathcal{A}$ is only able to passively observe them and does not get to learn the discrete logarithms: For one, g^s needs to be unrevealed (i.e., $\mathsf{st}_{\mathsf{ssk},\mathsf{sskid}} = \mathsf{fresh}$) in order for Test queries on 0-RTT keys derived from it to be allowed. This holds as a RevealSemiStaticKey query on a tested replayable stage would set the lost flag to true. At the same time, Corrupt queries do not reveal semi-static keys. Then, if the client side is tested, this side necessarily is honest and hence picks g^x, while, if the server side is tested, it must have an honest partnered client session, which means $\mathcal{A}$ cannot have modified the honestly picked ephemeral g^x on the way. Finally, both g^x and g^s are chosen independently of the other ephemeral and semi-static values (which $\mathcal{B}_3$, hence, can still select on its own), which in particular implies that $\mathcal{B}_3$ can detect a differing behavior of $\mathcal{A}$ in case, coincidentally, the same Diffie–Hellman shares are picked independently in another session.

Using the queries provided in the msPRF-ODH security game, $\mathcal{B}_3$ is moreover able to correctly compute keys from both further static secrets SS′ for server sessions using the same semi-static key g^s (without knowing s), as well as (once) the ephemeral secret ES′ in the involved client session (without knowing x). Hence, it can in particular correctly answer Reveal queries to any of these sessions.

Therefore, the advantage difference of $\mathcal{A}$ between Game B.2 and Game B.3 is, through $\mathcal{B}_3$'s output, transformed into a difference of outputting 1 in the two cases of the msPRF-ODH game and, hence, we can bound the former difference as

$$\mathsf{Adv}^{G_{B.2}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.3}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{msPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_3}.$$

Game B.4. In the next step, we replace the HKDF.Expand evaluations keyed with $\widetilde{\text{xSS}}$, in particular in the tested and matching sessions, by a (lazy-sampled) random function. This in particular results in the early-data handshake and application traffic keys tk_{ehs} and tk_{eapp}, the 0-RTT finished secret FS_0, and the expanded static secret mSS in the tested session being replaced by independent random values $\widetilde{tk_{ehs}}, \widetilde{tk_{eapp}}, \widetilde{\text{FS}_0}, \widetilde{\text{mSS}} \xleftarrow{\$} \{0,1\}^\lambda$.

We can turn any adversary $\mathcal{A}$ distinguishing this change (with non-negligible probability) into an adversary $\mathcal{B}_4$ against the PRF security of HKDF.Expand. Again, $\mathcal{B}_4$ acts as the Multi-Stage challenger for $\mathcal{A}$, this time using its PRF oracle for any HKDF.Expand evaluation under key $\widetilde{\text{xSS}}$, while performing evaluations under different keys on its own. In case the PRF oracle computes the real function, this simulation equals Game B.3; if the oracle computes a random function, it equals Game B.4. Moreover, the simulation is sound as $\widetilde{\text{xSS}}$ is an independent random value (due to the change in Game B.3) and hence chosen like the key in the PRF security game.

Thus, $\mathcal{B}_4$'s distinguishing advantage in the PRF game bounds the difference of $\mathcal{A}$ in the two games:

$$\mathsf{Adv}^{G_{B.3}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.4}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_4}.$$

With the change in Game B.4, both potentially tested keys $\widetilde{tk_{ehs}}$ and $\widetilde{tk_{eapp}}$ are now chosen independently at random. It remains to argue that the adversary cannot learn either value through a Reveal query.

Note first of all that a Reveal query is permissible only on non-partnered sessions. In order for such a session to derive the same key, it must hold the same key material and handshake messages entering the key derivation. For tk_{ehs}, the full session identifier $\mathsf{sid}_1 = (\mathtt{CH}, \mathtt{SC}^\dagger, \mathtt{SCRT}^\dagger)$ enters the key derivation, any non-partnered session hence necessarily derives a distinct key tk_{ehs} and a Reveal query thus does not affect the independent test session keys. For tk_{eapp}, messages $\mathtt{CCRT}_0^*$ and $\mathtt{CCV}_0^*$ are part of sid_2 but do not enter the key derivation (with no reason given for this design choice in the draft specification). In principle, the adversary $\mathcal{A}$ could hence modify these messages (e.g., by corrupting the client and sending a re-randomized $\mathtt{CCV}_0$ message) in order to have the server session holding $\mathrm{SS} = g^{xs}$ and hence also $\widetilde{tk_{eapp}}$ accept with a session identifier sid_2 different from the (tested) client session. This would allow $\mathcal{A}$ to issue a Reveal query on the server session and learn $\widetilde{tk_{eapp}}$. As we will see through the next and final game hop, a non-partnered session holding $\widetilde{tk_{eapp}}$ however implies a forged $\mathtt{ClientFinished}_0$ message (i.e., a MAC forgery), as that value fixes all messages in the session identifier sid_2 established.

Game B.5. Let Game B.5 be as before except that the challenger aborts if there accepts a session holding the same (replaced) extracted static secret $\widetilde{\mathrm{xSS}}$ and stage-2 key $\widetilde{tk_{eapp}}$, but which does not share the stage-2 session identifier sid_2 with the tested session. We show that in this case, the adversary made the server session of the two sessions accept with a forged MAC value in the $\mathtt{ClientFinished}_0$ message.

First of all observe that both sessions agree on the 0-RTT finished key FS_0 derived from $\widetilde{\mathrm{xSS}}$. The 0-RTT finished keys has moreover been replaced by a random value $\widetilde{\mathrm{FS}_0}$ in Game B.4). This enables the following reduction $\mathcal{B}_5$ to the EUF-CMA unforgeability of the HMAC scheme, showing that non-partnering while deriving the same key $\widetilde{tk_{eapp}}$ implies a successful MAC forgery.

In the reduction, $\mathcal{B}_5$ uses its MAC oracle to compute the $\mathtt{ClientFinished}_0$ message under key $\widetilde{\mathrm{FS}_0}$ (which it does not sample itself) over $H_3 = \mathsf{H}(\mathsf{sid}_2) = \mathsf{H}(\mathtt{CH} \| \mathtt{SC}^\dagger \| \mathtt{SCRT}^\dagger \| \mathtt{CR}^\dagger \| \mathtt{CCRT}_0^* \| \mathtt{CCV}_0^*)$. Now, if the described abort event above occurs, the server session holds some $\mathsf{sid}_2' = (\mathtt{CH}', \mathtt{SC}^{\dagger\prime}, \mathtt{CCRT}^{\dagger\prime}, \mathtt{CR}^{\dagger\prime}, \mathtt{CCRT}_0^{*\prime}, \mathtt{CCV}_0^{*\prime})$ which does not match the sid_2 on the client side. In order to accept the stage-2 key, the server hence must have obtained a MAC covering $\mathsf{H}(\mathsf{sid}_2')$. As there are no hash collisions due to Game 1, no client holding the same 0-RTT finished key $\widetilde{\mathrm{FS}_0}$ issued such MAC, and hence $\mathcal{B}_5$ did not query its MAC oracle on this value. We therefore let $\mathcal{B}_5$ output the $\mathtt{ClientFinished}_0$ MAC obtained by the server in the abort event as its valid EUF-CMA forgery, which allows us to bound the introduced difference in advantage as

$$\mathsf{Adv}^{G_{B.4}}_{\mathtt{draft\text{-}12\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.5}}_{\mathtt{draft\text{-}12\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_5}.$$

Finally, the Test query output is not only an independent random value, but the adversary can also not obtain this value as the result of a Reveal query to a non-partnered session (by Game B.5). Moreover, even if $\mathcal{A}$ replays the 0-RTT messages of the involved client session to further server sessions or injects the client's ephemeral share g^x in a differently crafted $\mathtt{ClientHello}$ message, this does not allow it to distinguish the real from the random key. In the former case, all sessions receiving the same client 0-RTT messages will be partnered with the test session (and hence those keys cannot be revealed). In the latter case, the keys tk_{ehs} and tk_{eapp} in those (non-partnered) sessions will be derived from the same key $\widetilde{\mathrm{xSS}}$ but with a distinct handshake hash (due to collisions being excluded by Game 1) and, hence, are independent random values themselves.

Therefore, b_{test} at this point remains unknown to $\mathcal{A}$ and thus

$$\mathsf{Adv}^{G_{B.4}}_{\mathtt{draft\text{-}12\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} \leq 0.$$

Case C. Stage 3–6: Test without Stage-3 Partner

This case can be proven as for the `draft-10` full (EC)DHE handshake (cf. Case A of the proof of Theorem 6.2 in Section 6.3), leveraging that acceptance without stage-3 partner requires a signature forgery (in `ServerCertificateVerify` or `ClientCertificateVerify`). Hence, the security bound established there still applies:

$$\mathsf{Adv}^{G_1,\text{test 3–6,test without partner}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_6}.$$

Case D. Stage 3–6: Test with Stage-3 Partner

For the last case, the following aspects need to be considered when adapting the `draft-10` full handshake proof (cf. Case B of the proof of Theorem 6.2 in Section 6.3) to the (full handshake part of the) `draft-12` 0-RTT handshake.

First, the ephemeral and static secrets $\mathrm{ES} = g^{xy}$ and $\mathrm{SS} = g^{xs}$ are now derived differently (instead of having ES = SS as in the full handshake). We hence encode the $\mathsf{snPRF\text{-}ODH}$ challenge (in Game B.2) in g^x and g^y only, replacing xES by a uniformly random element $\widetilde{\mathrm{xES}} \xleftarrow{\$} \{0,1\}^\lambda$ while the derivation SS remains unmodified. In particular, $\mathcal{A}$ is allowed to reveal the semi-static key g^s at any time via a $\mathsf{RevealSemiStaticKey}$, which we can respond to also in this step as the reduction picks s on its own.

Second, when replacing (in Game B.4) with random the master secret derived as $\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{mSS}, \widetilde{\mathrm{mES}})$, mSS is derived from $\mathrm{SS} = g^{xs}$ and hence may be known to the adversary through a $\mathsf{RevealSemiStaticKey}$ query on g^s. We thus cannot rely on the PRF security of $\mathsf{Extract}$ as in the full handshake proof. Instead, following the analysis of the core cryptographic protocol OPTLS by Krawczyk and Wee [KW16] underlying `draft-12`, we model $\mathsf{HKDF.Extract}$ as a strong extractor with $\widetilde{\mathrm{mES}}$ as entropy source and mSS as (public) seed (cf. Definition 3.8 in Section 3.2.3). The step of replacing MS with an independent random value $\widetilde{\mathrm{MS}} \xleftarrow{\$} \{0,1\}^\lambda$ can accordingly be bounded by the corresponding distinguishing advantage $\mathsf{Adv}^{\mathsf{st\text{-}Extract}}_{\mathsf{HKDF.Extract},\mathcal{B}_9}$ of an according reduction $\mathcal{B}_9$.

Third, as—in order to enable 0.5-RTT communication and in contrast to `draft-10`—the (stage-4) application traffic key tk_{app} is expanded from MS using only parts of the stage-4 session identifier sid_4, we need to ensure no non-partnered session derives the same key (as such session could otherwise be revealed). As for the similar situation with the early application traffic key tk_{eapp}, a non-partnered session deriving the same key must obtain a MAC within `ClientFinished` different from the one sent by the honest client. In the previous step, we replaced all keys derived from $\widetilde{\mathrm{MS}}$ by uniform random ones, so in particular is also the client finished key $\mathrm{FS_C}$ now chosen independently at random in the test session. We can hence bound the probability of the above event happening by the advantage $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_{11}}$ against the $\mathsf{EUF\text{-}CMA}$ security of HMAC of a reduction $\mathcal{B}_{11}$ similar to that for Game B.5 in Case B of this proof.

Considering these changes, the advantage bounds induced by the proof steps beyond that for the strong extractor and the added MAC forgery remain identical to those established for Case B of the proof of Theorem 6.2:

$$\begin{aligned}\mathsf{Adv}^{G_1,\text{test 3–6,test with partner}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq n_s \cdot \Big(&\mathsf{Adv}^{\mathsf{snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_7} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8} + \mathsf{Adv}^{\mathsf{st\text{-}Extract}}_{\mathsf{HKDF.Extract},\mathcal{B}_9} \\ &+ \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{10}} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_{11}}\Big).\end{aligned}$$

Combining the advantages bounds for all four proof cases yields the overall bound. □

7.6 Comparing the QUIC and TLS 1.3 0-RTT Handshakes

We emphasize two aspects here in which the TLS 1.3 design is superior to QUIC and strengthens the achievable (multi-stage) security both in terms of key independence and compositionality: First, it derives separate keys for the different purposes (in particular, tk_{ehs} and tk_{eapp} as well as tk_{hs} and tk_{app} for the encryption of (0-RTT resp. regular) handshake messages and data), enabling a cleaner key separation. Second, it establishes authenticity of the server's Diffie–Hellman share g^y through an explicit MAC (PSK-(EC)DHE 0-RTT) resp. signature ((EC)DHE 0-RTT) instead of through an authenticated encryption (under the 0-RTT key) in the data channel, rendering the security of one session key not relying on the secrecy of another. This in particular (and in contrast to the result for QUIC) enables an application of our multi-key composition theorem (Theorem 4.4) to the usage of the non-0-RTT external keys (tk_{app}, EMS, and RMS) in the record protocol, general symmetric protocols, and resumption, respectively, similar to the result for the TLS 1.3 `draft-10` full and PSK-based handshakes (see Section 6.6).

Conversely, QUIC in its original version Revision 20130620 achieves replay protection for the derived 0-RTT key on the key exchange level whereas TLS 1.3 in general does not (and hence, technically, TLS 1.3 satisfies only a weaker notion of security in that respect). Yet, recent TLS 1.3 drafts discuss that servers *should* implement additional anti-replay mechanisms, proposing, e.g., storing—as in QUIC—the nonce sent in the `ClientHello` or performing freshness checks via the age of session tickets and estimated round-trip times (see [Res18, Section 8]). While, as discussed in the beginning, anti-replay protection cannot prevent application-level replays of 0-RTT data through different secure channels, it seems that providing replay protection at the key exchange level emerges (again) as desirable security goal in practice. [Mac17]

Finally, the (abandoned) Diffie–Hellman-based and the (remaining) PSK-based 0-RTT handshakes in TLS 1.3 (as specified for `draft-12`, resp. `draft-14`) differ in the forward-secrecy guarantees they provide for 0-RTT keys, as already pointed out by Krawczyk on the TLS mailing list [Kra16a]. While in `draft-12` (EC)DHE 0-RTT those keys are forward secret (wrt. long-term (signing) key compromise) and succumb only to exposures of the semi-static key involved, no forward secrecy is provided in the PSK and PSK-(EC)DHE 0-RTT mode of `draft-14`. It is important to note, though, that pre-shared resumption secrets used in the PSK-based 0-RTT modes (treated as long-term secrets in our model) are usually much shorter-lived than public-key long-term signing keys, mitigating the effects of a compromise. More recent TLS 1.3 drafts in particular discuss the option to use resumption secrets only once, improving on the forward secrecy as well as anti-replay protection achieved [Res18, Section 8]. Still, pre-shared keys have to be stored safely by both the server and the client—a challenging task in practice, especially on the client side. Diffie–Hellman-based 0-RTT hence poses weaker requirements in that respect as the client here only has to store the public part of a semi-static key.

Chapter 8

The TLS 1.3 Protocol: A Formal Treatment of Key Confirmation

Summary. In this chapter we turn to a functional property of key exchange protocols beyond the classical notions of key secrecy and authentication: key confirmation. We present a game-based security model based on the classical Bellare–Rogaway model which provides the first rigorous formal treatment of key confirmation. We then analyze the TLS 1.3 full (EC)DHE handshake from version `draft-10` [Res15e] and show that it achieves desirable notions of key confirmation for both clients and servers. The results in this chapter are based on a work published at IEEE S&P 2016 [FGSW16].

8.1 Introduction

The seminal work of Bellare and Rogaway [BR94] (cf. Section 3.1) provided rigorous security definitions for the two core security goals of (authenticated) key exchange: key secrecy and (entity) authentication. An intuitively desirable security property that has so far escaped a comprehensive treatment is *key confirmation*: the idea that when a party accepts locally a key, it has the guarantee that some other party has precisely the same key. The property is often mentioned in scientific papers on the subject of key exchange [BPR00, Kra05, LLM07] but the typical reference for a definition is the "Handbook of Applied Cryptography" [MVO96, Definition 12.7] which describes key confirmation as the property

> "whereby one party is assured that a second (possibly unidentified) party actually has possession of a particular secret key."

Other references include the refinement proposed by Blake-Wilson and Menezes [BWM99a, BWM99b] who further distinguish between *explicit* key confirmation, where one party is assured that the other party holds the key, and *implicit* key confirmation, where the other party can compute the key. In another work, Blake-Wilson et al. [BWJM97] propose definitions for authenticated key agreement protocols with key confirmation which coincide with those of Bellare and Rogaway [BR94] for (entity) authentication. Accordingly, their notion aims at ensuring partnering rather than actual agreement on keys and furthermore ignores the inherent asymmetry in key confirmation, namely that one of the protocol participants must accept first and thus gets a different strength of confirmation guarantee than the one accepting last.

One may speculate that the reason for a lack of comprehensive and tailored definitions is that absence of key confirmation does not seem to open parties to attacks: a party may send messages encrypted with an (unconfirmed) key which no-one can decrypt. This may be a waste

of resources but not an obvious security risk. On the other hand, in some cases it might seem "clear" when a protocol has key confirmation. For example, protocols like TLS 1.2 [DR08] and EMV [EMV12] utilize the derived key during the execution of the protocol, so receiving a a message encrypted with the shared key provides key-confirmation assurances.

The latter level of informal understanding is also reflected by other folklore protocol transformations that can boost a secure key exchange protocol to also provide key confirmation guarantees. A popular proposal (which we refer to as "refresh-then-MAC") is to extend a key exchange protocol as follows: use the established key first to derive two additional keys; the first will be set as the session key whereas the latter will be used to compute a message authentication code (MAC) value over the transcript of the protocol so far. Exchanging valid MACs should then guarantee that parties have also locally computed the associated session key.

The status of key confirmation as an important security property is still unclear. On the one hand, some practitioners seem to be convinced that key confirmation messages (messages that use the key derived to ensure that the parties have agreed on the same key) do improve authentication; Adam Langley from Google, e.g., publicly proclaimed at the Real World Cryptography Workshop (RWC) 2014: "Key-confirmation messages are here to stay." [Lan14] Yet, others struggle to understand the benefits that key confirmation messages bring to protocols; see, e.g., the discussion on the IETF TLS mailing list on removing the confirmation message from the design of TLS [Lad14]. Nevertheless, many security protocol specifications name key confirmation as an explicit goal to achieve, including, e.g., the recommendations for key establishment schemes by NIST [BCRS13, BCRS09, HC09], or the draft specifications for the next TLS version 1.3 [Res18].

Security definitions. In this chapter, we propose security definitions (in Section 8.2) that aim to precisely capture the established intuition behind key confirmation. First, we note that we do not attempt to distinguish between explicit and implicit key confirmation: distinct computational interpretations to "has the key" and "can compute the key" seem difficult to provide. This follows the line of reasoning by Blake-Wilson and Menezes [BWM99a] who argue that "for all practical purposes, the assurances [of implicit and explicit key confirmation] are in fact the same," especially since one cannot guarantee that a party does not forget a key between its derivation and its first time usage.

Second, it is clear that key confirmation guarantees are asymmetric: the party that receives the last message obtains the stronger guarantees; such guarantees do not hold for the party which sends the last message since that message can be dropped by an adversary. Accordingly, we distinguish between *full key confirmation* and *almost-full key confirmation*. The former property guarantees that when a party accepts a key, there exists some other party that has already accepted precisely that key. The latter property ensures that when a party accepts a key, there is some session which, *if it accepts*, then it accepts the same key. Formalizing sound key confirmation notions turns out to be more challenging than one might expect given the common informal understanding. As we explain later, although we rely on compelling intuition, the definitions need to be carefully crafted to avoid some potential pitfalls which we outline.

A prime feature of our definitions is modularity. Following the approach of Blake-Wilson and Menezes [BWM99a] as well as NIST [BCRS13, BCRS09], we choose to disentangle key confirmation from the other security concerns specific to key exchange. In particular, our notion only ensures that there exists *some party* that accepted the same key, but does not guarantee that it is the expected communication partner. However, the desired property follows by combining key confirmation and implicit key authentication (i.e., classical key secrecy with mutually authenticating parties [BR94]): a protocol with both these properties has explicit key authentication. Informally, key confirmation can be interpreted as guaranteeing the lower bound

that "at least one other (unspecified) party holds the key" whereas implicit key authentication ensures the upper bound that "at most one (namely the expected) party holds the key." Together, the notions entail explicit key authentication: "exactly the expected party holds the key." Note that the definitional modularity also allows us to "swap" the key confirmation steps and property in and out, depending on the protocol's security requirements (as in the recommendations of NIST), and to independently argue about this additional security feature.

Application to TLS 1.3. We use the rigorous security models that we develop to shed light on the key-confirmation properties of the TLS 1.3 handshake design in `draft-10` [Res15e] in Section 8.3. As in previous TLS versions, TLS 1.3 `draft-10` leverages `Finished` messages essentially consisting of a message authentication code (MAC) computed over the transcript of the key exchange and sent both by the client and the server. It is hence not surprising that our analysis confirms that (the full, (EC)DHE handshake of) TLS 1.3 indeed achieves the strongest expectable key confirmation guarantees, i.e., full key confirmation for the server (which accepts after the client) and almost-full key confirmation for the client (which accepts first).

Perhaps surprisingly, we show that key confirmation does not (necessarily) rely on the `Finished` messages exchanged, but can actually be shown to hold even in a shortened variant of the full `draft-10` handshake which omits these messages. This becomes possible due to the `CertificateVerify` messages sent in the full `draft-10` handshake, which are essentially an online signature under the parties' long-term secret signing keys over (the hash of) all messages exchanged (i.e., the transcript or "session hash", as denoted in TLS 1.3).

This result deepens the understanding of the far-reaching security guarantees achievable with the session hash concept (originally introduced to counter the triple handshake attack [BDF+14] in TLS 1.2) and online signatures. While it might at first glance seem to open up a discussion of whether `Finished` messages become obsolete in presence of `CertificateVerify` messages already establishing key confirmation, we remark that TLS 1.3 specifies further handshake variants which omit the `CertificateVerify` messages for performance reasons and fully rely on the `Finished` messages for key confirmation as well as authentication. Furthermore, as we have seen in Section 7.5 for the (EC)DHE 0-RTT handshake in TLS 1.3 `draft-12`, changes in the key derivation to enable 0.5-RTT communication make `Finished` a necessary ingredient already for (multi-stage) key secrecy in later drafts.

Generic transformation. As explained above, one idea used to obtain key confirmation deployed in existing protocols is to somehow explicitly involve the key in the operations of the key-exchange protocol. This message plays a double role: on the one hand the MAC "ties" together the messages that belong to one session. On the other hand, the message is sent over the channel that is being established, or in other words, it is encrypted with the session key: receiving this message would therefore show that the other party already holds the key. It is by now well-known that unmitigated use of the session key (for either encrypting or MAC-ing) immediately destroys key secrecy, and better transformations have been proposed and used in protocol design. With precise definitions in place, we are in a position where these proposals can be accurately analyzed.

In addition to the above results presented in this thesis, we refer to [FGSW16] where we deploy our rigorous notions to analyze the popular "refresh-then-MAC" transformation (also recommended by NIST [BCRS13, HC09]) and confirm that the intuition behind the construction is indeed correct: when applied to a key-exchange protocol that ensures key secrecy the transformation yields a protocol which, in addition, also satisfies key confirmation (and preserves (implicit) authentication).

8.2 A Formal Model for Key Confirmation

In this section we develop and discuss our notions for key confirmation. To prepare the stage, we first augment the underlying Bellare–Rogaway security model in notation and define a generic security game we use for our definitions. We then design two security notions which capture strong forms of key confirmation, one that corresponds to the guarantees of the party that receives the last message in the protocol, and a second one for the party that sends this last message—per the discussion in the introduction we do not distinguish between explicit and implicit key confirmation. We give logical formulas that directly capture the basic intuition behind key confirmation and then turn the formulas into their associated security notions. As we explain, care needs to be taken to rule out superficially correct, but in fact misleading definitions.

8.2.1 Augmenting the Bellare–Rogaway Model

For capturing key confirmation, we build upon the basic Bellare–Rogaway model as a starting point, see Section 3.1 for its specification. Yet, we note that our formalism can in principle be lifted to multi-stage key exchange model from Chapter 4.

We begin by augmenting the Bellare–Rogaway model by a few, mostly syntactical, components. Most importantly, we introduce the notion of a *key-confirmation identifier* kcid, pivotal for our formalization of almost-full key confirmation. Essentially, once set in a session, the identifier kcid ensures that the session will eventually derive the same key as any other session with that identifier, even though the session has not accepted and the session identifier sid has not been set yet. In other words, one may interpret the setting of the key-confirmation identifier as stating that, upon receiving the partner's confirmation message, a session will have enough information to compute the (same) key. We elaborate on the choices for setting key-confirmation identifiers in a protocol further when introducing the notion of almost-full key confirmation that relies on them.

For syntactic convenience, in this chapter we consider participants in the protocol to to belong to either a set of clients $\mathcal{C}$ or a set of servers $\mathcal{S}$ (i.e., $\mathcal{U} = \mathcal{C} \cup \mathcal{S}$ for disjoint sets $\mathcal{C}$, $\mathcal{S}$). We furthermore assume clients to always act as the initiator in an execution and servers taking the role of the responder and hence sometimes interchangeably write $\mathsf{role} = \mathsf{client}$ instead of $\mathsf{role} = \mathsf{initiator}$ resp. $\mathsf{role} = \mathsf{server}$ instead of $\mathsf{role} = \mathsf{responder}$. As for the multi-stage model (cf. Section 4.2) we allow for unilateral authentication and that the communication partner of a session be unknown at the start and become "post-specified" during the session run.

The changes in the session list $\mathsf{List_S}$ compared to the plain Bellare–Rogaway model in Section 3.1.1 hence comprise:

- $\mathsf{pid} \in (\mathcal{U} \cup \{*\})$: the identity of the intended communication partner, where the distinct wildcard symbol '$*$' stands for "unknown identity" and can be set to a specific identity in $\mathcal{U}$ once by the protocol
- $\mathsf{kcid} \in \{0,1\}^* \cup \{\perp\}$: the key-confirmation identifier, initialized to $\perp$ and usually set at some point during the execution

We keep the set of oracle queries the adversary uses to interact with the protocol as set out in Section 3.1.2 for the Bellare–Rogaway model, except for the now superfluous Test query. For later reference, we record in an (initially empty) set Corr the identities corrupted via a $\mathsf{Corrupt}$ query; all other parties in $\mathcal{U} \setminus \mathsf{Corr}$ are called honest or uncorrupted.

We furthermore introduce the following abbreviations to simplify formulas below: We abbreviate session labels label as ℓ for brevity. We define two predicates $\mathsf{partners}$ and $\mathsf{samekey}$

which on input two session labels ℓ, ℓ' evaluate to true if and only if the two sessions indicated are partnered (i.e., $\ell.\mathsf{sid} = \ell'.\mathsf{sid}$ and $\ell \neq \ell'$), resp. hold the same key (i.e., $\ell.\mathsf{key} = \ell'.\mathsf{key}$).

8.2.2 A Generic Security Game

It will be convenient to formalize key confirmation via a "success" predicate Pred in a generic security game $G^{\mathsf{Pred}}_{\mathsf{KE},\mathcal{A}}$ for some key exchange protocol KE and a PPT adversary $\mathcal{A}$. In the following we will instantiate the predicate for our different key confirmation notions, but note that other properties can be captured via such predicates as well (for example Match security, see [FGSW16]).

Definition 8.1 (Generic Pred security game)**.** *Let* Pred *be an abstract predicate,* KE *be a key exchange protocol, and* $\mathcal{A}$ *a PPT adversary* $\mathcal{A}$ *interacting with* KE *via the queries defined in Section 3.1.2 in the following generic security game* $G^{\mathsf{Pred}}_{\mathsf{KE},\mathcal{A}}$*:*

Setup. *The challenger generates long-term public/private-key pairs for each participant* $U \in \mathcal{U}$*.*

Query. *The adversary* $\mathcal{A}$ *receives the generated public keys and has access to the queries* $\mathsf{NewSession}$*,* Send*,* Reveal*, and* $\mathsf{Corrupt}$*.*

Stop. *At some point, the adversary stops with no output.*

We say that $\mathcal{A}$ *wins the game, denoted by* $G^{\mathsf{Pred}}_{\mathsf{KE},\mathcal{A}} = 1$*, if it violates the predicate* Pred *in the sense that* Pred *evaluates to* false *on the final execution state of the game. We say* KE *provides* Pred security *if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{Pred}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{Pred}}_{\mathsf{KE},\mathcal{A}} = 1\right].$$

8.2.3 Full Key Confirmation

We are now ready to present our notions for key confirmation and first treat the simpler case of *full* key confirmation. These are the guarantees obtained by the party that receives the last message of the protocol: the protocol ensures full key confirmation (for that party) if, when it receives this last message (and therefore accepts the locally derived key), it has the guarantee that there is a (partnered) session of the protocol that has accepted precisely the same key.

Since a protocol cannot achieve the full confirmation property for all sessions simultaneously—in each pair of sessions one party has to finish first—it is convenient to restrict the sessions under considerations to some subset. Since a session is fully described by its label ℓ, including for example the identity of the party running the session, we usually identify the sessions according to their label ℓ which should belong to some set $\mathcal{L}$. Slightly overloading notation (but extending our predicate-based notions above in a natural way) a label ℓ belongs to $\mathcal{L}$ if $\mathcal{L}(\ell) = \mathsf{true}$. For example $\mathcal{L}(\ell) = [\ell.\mathsf{id} \notin \mathsf{Corr}]$ would comprise all honest parties' sessions. Analogously, we conveniently reuse the identity sets $\mathcal{C}$ for client and $\mathcal{S}$ for server session labels by defining $\mathcal{C}(\ell) = [\ell.\mathsf{role} = \mathsf{client}]$ resp. $\mathcal{S}(\ell) = [\ell.\mathsf{role} = \mathsf{server}]$.

In the definition of the full key confirmation predicate below we abstractly speak of subsets $\mathcal{L}$ and $\mathcal{L}'$ of all labels. The predicate stipulates that for each accepting session with a label ℓ from $\mathcal{L}$, where the partner is neither corrupt nor unauthenticated (in which case the adversary could impersonate the partner), there exists another session with a label ℓ' from $\mathcal{L}'$ such that this session also accepts the same key. Note that we do not demand that the session ℓ' is actually held by the intended partner specified by $\ell.\mathsf{pid}$ (which is captured as a distinct modular property within Match security), but only that it is partnered according to the session identifiers. This conveniently allows combining (full) key confirmation with other sid-based security notions, e.g.,

to achieve authentication and partnering properties when coupled with (implicit) authentication resp. Match security or to achieve explicit key authentication when linked with key secrecy.[42]

As a final technical remark, key confirmation can only be expected for sessions that communicate with a distinct, uncorrupted party, as we cannot reason about an adversarially controlled session deriving certain values or holding the same key. This is reflected in both the definitions of full and almost-full key confirmation by demanding that $\ell.\mathsf{pid} \notin \mathsf{Corr} \cup \{*\}$.

Definition 8.2 (Full key confirmation predicate)**.** *The predicate* $\mathsf{FullConf}(\mathcal{L}, \mathcal{L}')$ *that defines full key confirmation is the following:*

$$\begin{aligned}&\forall \ell \in \mathcal{L} :: \ [\ell.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted} \wedge \ell.\mathsf{pid} \notin \mathsf{Corr} \cup \{*\}] \\ &\quad \implies \ [\exists \ell' \in \mathcal{L}' :: \ (\ell'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted} \wedge \mathsf{partners}(\ell, \ell') \wedge \mathsf{samekey}(\ell, \ell'))].\end{aligned}$$

Note that $\mathsf{partners}(\ell, \ell')$ ensures that $\ell \neq \ell'$.

A protocol offers full key confirmation if no efficient adversary can make the predicate false, except with negligible probability.

Definition 8.3 (Full key confirmation)**.** *A key exchange protocol* KE *provides full* $(\mathcal{L}, \mathcal{L}')$*-key confirmation if for all PPT adversaries* $\mathcal{A}$ *the following advantage function for the generic security game from Definition 8.1 is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{FullConf}(\mathcal{L},\mathcal{L}')}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{FullConf}(\mathcal{L},\mathcal{L}')}_{\mathsf{KE},\mathcal{A}} = 1\right].$$

Note that the notion above implicitly captures the time-critical aspect that some other session already holds the same key at the point in time when a party accepts. Whereas the predicate FullConf is evaluated on the final execution state, thus not allowing to distinguish between sessions ℓ for which the paired session ℓ' existed *before* or only *after* session ℓ accepted, the quantification over all adversaries $\mathcal{A}$ rules out the case that there has not been such a session ℓ' before. That is, assume that (any of the possibly multiple paired sessions) ℓ' only accepted after ℓ in an execution of some adversary $\mathcal{A}$. Then one can imagine a pruned version of the adversary which stops immediately after ℓ has accepted, say, simply by picking a random stop point in the execution. If $\mathcal{A}$ triggers the event that any paired session ℓ' existed only afterwards with non-negligible probability, then the pruned version of $\mathcal{A}$ (which is contained in the quantification) would break full key confirmation as above.

8.2.4 Almost-Full Key Confirmation

We now turn to the guarantees that key confirmation can offer to the party that sends the last message in a protocol and which therefore has no guarantee that its intended partner accepts (since the adversary may simply drop that last message).

A false start. To understand the subtleties involved in designing a definition for this case, we first explore a possible notion which, although intuitively appealing, has important shortcomings.

Intuitively, the best guarantee for the party that sends the last message (and accepts) is that there is some other session which, if it eventually accepts, will have accepted the same key (and is partnered). This intuition is captured by the following formula:

$$\begin{aligned}&\forall \ell \in \mathcal{L} :: \ [\ell.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted} \wedge \ell.\mathsf{pid} \notin \mathsf{Corr} \cup \{*\}] \implies \\ &\quad \left[\exists \ell' \in \mathcal{L}' :: \ (\ell'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted} \implies \mathsf{partners}(\ell, \ell') \wedge \mathsf{samekey}(\ell, \ell'))\right].\end{aligned}$$

[42] Consulting once more the "Handbook of Applied Cryptography" [MVO96], the authors there also treat the properties separately, and define explicit key authentication as the combination of key confirmation with (implicit) key authentication (where the latter comprises authenticity and key secrecy).

It turns out that this notion is too weak. The problem is that the predicate is satisfied whenever there is some session ℓ' that has not accepted. To understand why this is the case, consider the negation of the above predicate (which an adversary that attempts to break the property must ensure it evaluates to true).

$$\exists \ell \in \mathcal{L} :: \ \ell.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted} \wedge \ell.\mathsf{pid} \notin \mathsf{Corr} \cup \{*\}$$
$$\wedge \left[\forall \ell' \in \mathcal{L}' :: \ (\ell'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted} \wedge (\neg \mathsf{partners}(\ell, \ell') \vee \neg \mathsf{samekey}(\ell, \ell')))\right].$$

Note that to make the predicate true, the adversary has to ensure that all sessions accept. As soon as a single session ℓ' rejects, the formula cannot be satisfied anymore and the adversary loses. This is clearly too restrictive since at least for sessions unrelated to ℓ, the adversary should not be required to make them accept. To fix the definition, we have to take additional information into account to characterize sessions that will compute the same key as ℓ.

The right definition. We define the notion of *almost-full key confirmation* based on sessions which are waiting to receive the final message. Note that these are sessions which still lack some information to express the full session identifiers; we thus revert to *key-confirmation identifiers* for a weaker type of partnering. Almost-full key confirmation then ensures that the identified session holding the same key-confirmation identifier indeed accepts with the same key and is partnered (if it eventually accepts at all). This essentially captures the previous, fallen-short intuition of having another session that, if it eventually accepts, is partnered and derives the same key, but restricts this requirement to sessions agreeing on the same key-confirmation identifier.

Definition 8.4 (Almost-full key confirmation predicate)**.** *The predicate* $\mathsf{AlmostConf}(\mathcal{L}, \mathcal{L}')$ *that defines almost-full key confirmation is the following:*

$$\forall \ell \in \mathcal{L} :: \ [\ell.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted} \wedge \ell.\mathsf{pid} \notin \mathsf{Corr} \cup \{*\}] \implies$$
$$[\exists \ell' \in \mathcal{L}' :: \ (\ell.\mathsf{kcid} = \ell'.\mathsf{kcid} \wedge (\ell'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted} \implies \mathsf{partners}(\ell, \ell') \wedge \mathsf{samekey}(\ell, \ell')))].$$

Note that in contrast to our original formalization [FGSW16] we here also include the partnering condition $\mathsf{partners}(\ell, \ell')$ in the definition of almost-full key confirmation. This is in order to unify full and almost-full key confirmation with respect to demanding existence of a session that is partnered and derives the same key. As for full key confirmation, including the $\mathsf{partners}$ condition facilitates combining (almost-full) key confirmation with other sid-based security notions. We remark that this modification does neither affect the relation between full and almost-full key confirmation established below nor does it affect our later proof for key confirmation in TLS 1.3. As we will see, establishing the same keys naturally coincides with partnering there, as we also assert for full key confirmation.

Key-confirmation identifier binding. So far, key-confirmation identifiers, on which the definition of the almost-full key confirmation predicate are based upon, are not bound to the actual session identifiers or to keys. In order to give them practical meaning, we need to establish links to the notion of partnering as well as the derived keys.

First, it is natural to require that whenever two sessions are partnered, they in particular agree on the key-confirmation identifier.[43] More importantly, key-confirmation identifiers are supposed to capture the idea that, whenever two sessions accept and hold the same key-confirmation

[43] We note that, beyond this connection, we do not require any particular properties (e.g., concerning authentication) from session identifiers in the context of key confirmation. These aspects can be modularly captured through Match security and a definition of (implicit) authentication.

identifier, they also derive the same key (and are partnered). We formalize these concepts by defining the predicate $\mathsf{KCIDbind}$ which returns true if and only if all of the following conditions holds.

1. For all sessions ℓ, ℓ' with $\mathsf{partners}(\ell, \ell') = \mathsf{true}$, it holds that $\ell.\mathsf{kcid} = \ell'.\mathsf{kcid}$, i.e., partnered sessions agree on the same key-confirmation identifier.

2. For all sessions ℓ, ℓ' with $\ell.\mathsf{kcid} = \ell'.\mathsf{kcid}$ and $\ell.\mathsf{st}_{\mathsf{exec}} = \ell'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}$, it holds that $\mathsf{partners}(\ell, \ell') = \mathsf{true}$ and $\mathsf{samekey}(\ell, \ell') = \mathsf{true}$, i.e., sessions with the same key-confirmation identifier, upon acceptance, will be partnered and derive the same key.

Definition 8.5 (Key-confirmation identifier binding)**.** *A key exchange protocol* KE *provides key-confirmation identifier binding if for all PPT adversaries* $\mathcal{A}$ *the following advantage function for the generic security game from Definition 8.1 is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{KCIDbind}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{KCIDbind}}_{\mathsf{KE},\mathcal{A}} = 1\right].$$

Defining almost-full key confirmation. We are now ready to define almost-full key confirmation.

Definition 8.6 (Almost-full key confirmation)**.** *A key exchange protocol* KE *provides almost-full* $(\mathcal{L}, \mathcal{L}')$*-key confirmation if it satisfies key-confirmation identifier binding and for all PPT adversaries* $\mathcal{A}$ *the following advantage function for the generic security game from Definition 8.1 is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{AlmostConf}(\mathcal{L},\mathcal{L}')}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{AlmostConf}(\mathcal{L},\mathcal{L}')}_{\mathsf{KE},\mathcal{A}} = 1\right].$$

To elaborate why this formalization of almost-full key confirmation captures the right property, let us first again consider the negation of the predicate $\mathsf{AlmostConf}$ (i.e., the formula the adversary needs to make evaluate to true):

$$\begin{aligned}&\exists \ell \in \mathcal{L} :: \ \ell.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted} \wedge \ell.\mathsf{pid} \notin \mathsf{Corr} \cup \{*\} \\ &\wedge \left[\forall \ell' \in \mathcal{L}' :: \ \ell.\mathsf{kcid} \neq \ell'.\mathsf{kcid} \vee (\ell'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted} \wedge (\neg\mathsf{partners}(\ell, \ell') \vee \neg\mathsf{samekey}(\ell, \ell')))\right].\end{aligned}$$

First of all note that the part $\ell.\mathsf{kcid} \neq \ell'.\mathsf{kcid}$ formalizes that a protocol cannot choose to have unique key-confirmation identifiers per session, e.g., by setting the identifier to some local random value. This is so as this would trivially mean that for any session $\ell \in \mathcal{L}$ that an adversary initiates, all other sessions $\ell' \in \mathcal{L}'$ have non-matching key-confirmation identifiers, so the adversary immediately wins.

When a protocol instead lets every session $\ell \in \mathcal{L}$ accept with a kcid that matches the one of some session $\ell' \in \mathcal{L}'$, this allows the adversary to focus on such matching sessions. In contrast to the initial false-start formalization, the adversary can in particular let sessions reject that do not hold the same key-confirmation identifier as ℓ.

Finally, a trivial way to achieve almost-full key confirmation is for a protocol to set kcid to the same (e.g., empty) value for every session. Note that key-confirmation identifier binding then in turn requires that every session accepts with the same key. Although this rightly appears to be unreasonable (as it contradicts key secrecy), it is consistent from the perspective of key confirmation: If every session derives the same key, every session is trivially assured that, if there is another accepting session, it will hold the same key.

Choosing a key-confirmation identifier. A natural question arising from the definition of almost-full key confirmation is how to set the key-confirmation identifiers for a specific protocol. As for the regular session identifiers and their use within the freshness condition for defining key secrecy, there is an interplay between the security notion (key secrecy, resp. almost-full key confirmation) and the soundness requirements for the identifiers (Match security, resp. key-confirmation identifier binding).

On the one hand, to achieve almost-full key confirmation, a protocol has to couple up any accepting session (in $\mathcal{L}$) with a session (in $\mathcal{L}'$) holding the same key-confirmation identifier kcid. As already discussed, this in particular prevents using a unique kcid value per session. On the other hand, choosing the same key-confirmation identifier (e.g., an empty kcid) for every session, by key-confirmation identifier binding, implies that every session must derive the same key. As this in particular contradicts key secrecy, it is also not a viable option for any reasonable key exchange protocol.

Therefore a protocol needs to balance out the choice for setting key-confirmation identifiers between these two extremes. From a practical point of view, the key-confirmation identifier would intuitively comprise as much of the session identifier such that, together with the last protocol message, it fully determines the derived key. In some cases, even the actual key might already be computable (and hence serve as a "trivial" key-confirmation identifier) before the last message is received. The generic transformation based on an additional exchange of MACs discussed in [FGSW16] is such an example. In many practical protocols (that intuitively achieve almost-full key confirmation), however, the last message(s) will substantially contribute to the key and, hence, only partial information is available when setting (and hence captured in) the key-confirmation identifier. This is for example the case in TLS 1.3 (cf. Section 8.3 for our detailed analysis), where the key is derived from a hash over all messages, including some of the client's last messages. Therefore, when the client in TLS 1.3 accepts, the server does not know these messages yet and cannot have set kcid based on the key. Instead, we need to leverage the already exchanged part of the session identifier as key-confirmation identifier, which then fixes a unique key together with the client's last messages. This motivates why we chose to capture "agreement on the same key up to receipt of the last message" using a generic identifier string rather than relying on a particular protocol value or a partial communication transcript.

8.2.5 Relationship

We now take a look at the relationship between full and almost-full key confirmation and see why the former implies the latter (given key-confirmation identifier binding).

Theorem 8.7. *Let* KE *be key exchange protocol that provides full* $(\mathcal{L}, \mathcal{L}')$*-key confirmation as well as key-confirmation identifier binding. Then* KE *also provides almost-full* $(\mathcal{L}, \mathcal{L}')$*-key confirmation.*

Proof. We need to show that, for any session $\ell \in \mathcal{L}$ that accepts ($\ell.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}$) with a distinct, uncorrupted partner ($\ell.\mathsf{pid} \notin \mathsf{Corr} \cup \{*\}$), there exists a session $\ell' \in \mathcal{L}'$ which

(a) shares the same key-confirmation identifier ($\ell.\mathsf{kcid} = \ell'.\mathsf{kcid}$) and

(b) on acceptance is partnered and derives the same key ($\ell'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted} \implies \mathsf{partners}(\ell, \ell') \wedge \mathsf{samekey}(\ell, \ell')$).

First, observe that by full $(\mathcal{L}, \mathcal{L}')$-key confirmation, for any such session $\ell \in \mathcal{L}$ there exists a session $\ell' \in \mathcal{L}'$ that accepted ($\ell'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}$), is partnered with ℓ ($\mathsf{partners}(\ell, \ell')$), and holds the same key ($\mathsf{samekey}(\ell, \ell')$). Through the two latter assertions, this session ℓ' in particular

satisfies (b), i.e., partnering and derivation of the same key (on acceptance). Due to key-confirmation identifier binding, partnering implies that ℓ' furthermore also shares the same key-confirmation identifier (i.e., $\ell.\mathsf{kcid} = \ell'.\mathsf{kcid}$), satisfying (a). □

Match security vs. key confirmation. On a more distant relation, let us note that Match security and key confirmation (both full and almost-full) are independent notions (i.e., a protocol can provide either one without providing the other one). On the one hand, setting $\mathsf{sid} = \mathsf{kcid}$ to be a unique string per session trivially lets the protocol satisfy Match security, but renders full and almost-full key confirmation unachievable. On the other hand, having all sessions use the same (arbitrary) identifiers sid and kcid and derive the same key key trivially satisfies full and almost-full key confirmation, but violates Match security (due to more than two sessions being partnered with each other).

Match security thus rather constitutes a soundness counterpart to the freshness conditions used for key secrecy than being related to key confirmation.

8.2.6 Confirmation Guarantees for Unauthenticated Peers

Informal definitions of key confirmation usually demand that another, even *possibly unidentified* party holds the same key (e.g., the "Handbook of Applied Cryptography" [MVO96, Definition 12.7]). Note that our notions of full and almost-full key confirmation originally guarantee that for sessions which communicate with an *identified* (and uncorrupted) partner ($\ell.\mathsf{pid} \notin \mathsf{Corr} \cup \{*\}$) there is another session which (eventually) holds the same key.

The extension of our notions to unauthenticated peers turn out to hold trivially by correctness(-like) properties. For this, first note that key confirmation guarantees for sessions with unauthenticated peers can only be provided if the actual communication partner is indeed honest. This is so since no confirmation model can ensure that an adversarially-controlled session derives a session key at some point.

In the case of full key confirmation, any extension to unauthenticated peers (i.e., dropping the prerequisite $\ell.\mathsf{pid} \notin \mathsf{Corr} \cup \{*\}$ for the session ℓ in question) would hence need to condition on the existence of a partnered (honest) session to the session ℓ, e.g., adding the requirement $\exists \ell'' \in \mathcal{L}' :: \mathsf{partners}(\ell, \ell'')$ to the full key confirmation predicate. But then any (correct) key exchange protocol provides this kind of "unauthenticated" full key-confirmation anyway: Correctness demands that partnered sessions (i.e., with the adversary only passively relaying messages) derive the same key. This means that such a session ℓ'' already serves as a partnered session holding the same key according to our original definition.

In the case of almost-full key confirmation, guarantees for unauthenticated peers would again need to be conditioned on an existing session sharing the same key confirmation identifier (e.g., swapping in the requirement $\exists \ell'' \in \mathcal{L}' :: \ell.\mathsf{kcid} = \ell''.\mathsf{kcid}$ for $\ell.\mathsf{pid} \notin \mathsf{Corr} \cup \{*\}$). But then again, the natural requirement of key-confirmation identifier binding (being part of the almost-full key confirmation property) already satisfies almost-full key-confirmation for such cases trivially: It ensures that two sessions sharing the same key-confirmation identifier, upon acceptance, will derive the same key. Thus, session ℓ'' again serves as the desired partnered session according to out current notion.

In conclusion, obtaining no advantage from definitions encompassing unauthenticated peers, we focus on the key confirmation guarantees attainable when communicating with an authenticated partner.

8.3 Key Confirmation in TLS 1.3

We now apply our model to investigate the key confirmation properties provided by the upcoming version of the Transport Layer Security protocol, TLS 1.3. More precisely, we consider the TLS 1.3 draft `draft-10` [Res15e] and its full (EC)DHE handshake. In this chapter we focus on key confirmation, see Section 6.3 for the analysis of (multi-stage) key secrecy and Match security of the full (EC)DHE handshake of `draft-10`. We also do not treat key confirmation in the other (PSK-based and 0-RTT) handshake modes specified in `draft-10` or extensions added in follow-up drafts like post-handshake messages, but comment on later modifications along the way.

We refer to Section 6.2 for a detailed description of the TLS 1.3 `draft-10` full (EC)DHE handshake mode, including the relevant notation and an illustration of the handshake protocol and key schedule in Figure 6.1 (on page 61). In our treatment of key confirmation, we focus on the main application traffic key tk_{app} derived in the TLS 1.3 handshake (the stage-2 key in the multi-stage description of Section 6.2), which, we recall, is derived from the (hashed) session transcript $H_{\text{sess}} \leftarrow \mathsf{H}(\mathtt{CH}\| \ldots \|\mathtt{CCV}^*)$ up to the client's `CertificateVerify` message. We adopt the corresponding session identifier (sid_2 in Section 6.2), $\mathsf{sid} = (\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}, \mathtt{EE}, \mathtt{SC}^*, \mathtt{SCRT}^*, \mathtt{CR}^*, \mathtt{SCV}^*, \mathtt{CCRT}^*, \mathtt{CCV}^*)$. We define the key-confirmation identifier to be set to $\mathsf{kcid} = \mathtt{ClientHello}\| \ldots \|\mathtt{ServerCertificateVerify}$ by the server on sending its `ServerCertificateVerify` message and by the client on receiving that message. All other keys are considered to be protocol-internal, yet we do consider their usage, e.g., encryption of handshake messages under the intermediate handshake traffic key.

As in previous versions, TLS 1.3 in `draft-10` employs `Finished` messages (sent both by the client and the server), which are essentially MAC values computed over the (hashed) transcript (excluding the `Finished` messages), the so-called "session hash", in order to "provide[] key confirmation" [Res15e, p. 31]. Importantly, in contrast to previous TLS versions, the `Finished` messages do *not* rely on the derived session key, but are issued under a separate finished secret FS derived through a key derivation function from one of the secret Diffie–Hellman values established within the key exchange. In that sense, `draft-10` essentially follows the popular paradigm we discussed earlier (and treat in more detail in [FGSW16]) to exchange MACs over the transcript (or the session identifiers) in order to achieve key confirmation.

8.3.1 Key Confirmation without Finished Messages

Interestingly, we can however actually show that already a shortened variant of the `draft-10` handshake *without* the `ClientFinished` and `ServerFinished` messages (which we denote as `draft-10-nf`) can provide the same strongest form of key confirmation expectable. That is, in the mutually authenticating handshake the server is assured that the client already accepted with the same key at the time the server accepts while the client is assured that, if the server later accepts, it will do so with the same key. In contrast, in the unilaterally authenticating handshake—as expected—only the client is guaranteed (full) key confirmation.[44] We will first elaborate in detail how to prove key confirmation for the shortened `draft-10-nf` handshake and then demonstrate in Section 8.3.2 how this result can easily be adapted to the actual `draft-10` handshake.

More formally, we show that the mutually authenticating `draft-10-nf` handshake (short: `draft-10-nf-m`) achieves full $(\mathcal{S},\mathcal{C})$-key confirmation and almost-full $(\mathcal{C},\mathcal{S})$-key confirmation and that the unilaterally authenticating `draft-10-nf` handshake (`draft-10-nf-u`) provides full $(\mathcal{C},\mathcal{S})$-key confirmation. While we analyze both authentication variants of the handshake separately,

[44] Observe that, for unilateral authentication, in contrast to `draft-10` the server sends the last protocol message in the shortened `draft-10-nf` variant due to the omitted finished messages.

we remark that both results also hold when handshakes are allowed to run concurrently with mutual and with unilateral in our model, as the message flow is unique for each authentication variant which allows to tell the according sessions apart.

For both handshake variants our proofs rely on the `ClientCertificateVerify` respectively `ServerCertificateVerify` message exchanged, which essentially is a signature over the almost-complete transcript and can, hence, intuitively be seen as a signature-based analogue of the generic MAC-based transform discussed earlier and in [FGSW16]. Notably, in contrast to the requirements for the (multi-stage) key secrecy of the `draft-10` full (EC)DHE handshake in Section 6.3, we need to rely on the strong unforgeability (SUF-CMA) instead of existential unforgeability (EUF-CMA) of the deployed signature scheme here (we remark that this change corrects the original proof in [FGSW16]). Essentially, the signature in the `CertificateVerify` message also entering the key derivation requires its uniqueness in terms of unforgeability; we discuss the detailed technical reason in the proofs below.

Theorem 8.8. *The TLS 1.3* `draft-10-nf-m` *full (EC)DHE handshake for mutual authentication without finished messages satisfies full $(\mathcal{S},\mathcal{C})$-key confirmation and almost-full $(\mathcal{C},\mathcal{S})$-key confirmation. Formally, for any efficient adversary $\mathcal{A}$ against full $(\mathcal{S},\mathcal{C})$-key confirmation, resp. almost-full $(\mathcal{C},\mathcal{S})$-key confirmation, there exist efficient algorithms $\mathcal{B}_1$, $\mathcal{B}_2$ such that*

$$\mathsf{Adv}^{\mathsf{Pred}}_{\texttt{draft-10-nf-m},\mathcal{A}} \leq n_s^2 \cdot 2^{-|nonce|} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_u \cdot \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2},$$

for $\mathsf{Pred} = \mathsf{FullConf}(\mathcal{S},\mathcal{C})$*, resp.* $\mathsf{Pred} = \mathsf{AlmostConf}(\mathcal{C},\mathcal{S})$*, where n_u is the maximum number of users, n_s is the maximum number of sessions, and $|nonce| = 256$ is the bit-length of the nonces.*

Proof. We show that, for a server session, the `ClientCertificateVerify` message of an honest client suffices as assurance that this client has accepted with the same key when the server session accepts. In turn, for a client session, the `ServerCertificateVerify` message of an honest server ensures that this server agrees on the key-confirmation identifier kcid and will, if it accepts, derive the same key. To this extend we modify the original $G^{\mathsf{Pred}}_{\texttt{draft-10-nf-m},\mathcal{A}}$ game in three steps, showing that the advantage difference of adversary $\mathcal{A}$ can each time be bounded by the advantage of breaking the security of some TLS 1.3 component, finally reaching a game where the advantage of $\mathcal{A}$ is 0.

First, we consider the modified game $G^{\mathsf{Pred}'}_{\texttt{draft-10-nf-m},\mathcal{A}}$ which is as before, except it aborts (and outputs 0) if, during the execution, any two honest sessions choose the same nonce (r_c or r_s). The probability that the game aborts can be bounded from above by $n_s^2 \cdot 2^{-|\mathrm{nonce}|}$ where n_s is the maximum number of sessions and $|\mathrm{nonce}|$ is the nonces' bit-length. Therefore, denoting by $\mathsf{Adv}^{\mathsf{Pred}'}_{\texttt{draft-10-nf-m},\mathcal{A}}$ the advantage of $\mathcal{A}$ in $G^{\mathsf{Pred}'}_{\texttt{draft-10-nf-m},\mathcal{A}}$,

$$\mathsf{Adv}^{\mathsf{Pred}}_{\texttt{draft-10-nf-m},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{Pred}'}_{\texttt{draft-10-nf-m},\mathcal{A}} + n_s^2 \cdot 2^{-|\mathrm{nonce}|}.$$

Second, we switch to $G^{\mathsf{Pred}''}_{\texttt{draft-10-nf-m},\mathcal{A}}$ in which we additionally abort if, during the execution, any two honest sessions compute the same hash value for two different inputs to the hash function H. We can bound the probability that the game aborts for this reason by the advantage $\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}$ of an adversary $\mathcal{B}_1$ against the collision resistance of H. For this purpose, $\mathcal{B}_1$ simply simulates $G^{\mathsf{Pred}''}_{\texttt{draft-10-nf-m},\mathcal{A}}$ for $\mathcal{A}$ on its own and outputs the two colliding inputs when they occur during the simulation, perfectly simulating the experiment up to this point and always winning when the modified experiment aborts. Hence we have that

$$\mathsf{Adv}^{\mathsf{Pred}'}_{\texttt{draft-10-nf-m},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{Pred}''}_{\texttt{draft-10-nf-m},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

Third, we consider $G^{\mathsf{Pred}'''}_{\texttt{draft-10-nf-m},\mathcal{A}}$, which, in addition, aborts whenever a simulated server or client session obtains, within the `ClientCertificateVerify`, resp. `ServerCertificateVerify`,

message, a valid signature (under the public key of some non-corrupted client, resp. server, $U \in \mathcal{U}$) which was not output by any honest client resp. server session.

We can bound the probability of this abort by the advantage $\mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2}$ of an adversary $\mathcal{B}_2$ against the strong unforgeability of the deployed signature scheme Sig, simulating the experiment for $\mathcal{A}$ as follows. Initially, $\mathcal{B}_2$ randomly chooses a party U among the at most n_u parties, associating the challenge public key pk^* with it, and generates the long-term key for all other parties $V \in \mathcal{U} \setminus \{U\}$. During the simulation, $\mathcal{B}_2$ then uses its signing oracle whenever a signature needs to be computed under the secret key of U. When a simulated session obtains (within `SCV` or `CCV`) a valid signature on the expected hashed transcript that no other honest session has output, $\mathcal{B}_2$ aborts the experiment and outputs that signature (together with the hashed transcript) as its forgery. As no session output the signature in question, $\mathcal{B}_2$ did not query its oracle on the hashed transcript value resulting in that signature, which hence constitutes a valid SUF-CMA forgery. Note that $\mathcal{B}_2$ might have obtained a different signature on the same hashed transcript in a partnered session (i.e., a signature on the same message but not the same signature was output by an honest session), which is why the obtained signature may not constitute an EUF-CMA forgery and we thus need to rely on SUF-CMA security.

The overall bound is then conditioned on $\mathcal{B}_2$ correctly guessing the (non-corrupted) identity U (among the at most n_u identities) under whose public key pk^* the obtained signature verifies:

$$\mathsf{Adv}^{\mathsf{Pred}''}_{\texttt{draft-10-nf-m},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{Pred}'''}_{\texttt{draft-10-nf-m},\mathcal{A}} + n_u \cdot \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2}.$$

Finally, we can now separately argue along the lines of the predicates $\mathsf{Pred} = \mathsf{FullConf}(\mathcal{S},\mathcal{C})$ and $\mathsf{Pred} = \mathsf{AlmostConf}(\mathcal{C},\mathcal{S})$ as well as $\mathsf{Pred} = \mathsf{KCIDbind}$ (which is required as part of almost-full key confirmation):

- For $\mathsf{Pred} = \mathsf{FullConf}(\mathcal{S},\mathcal{C})$, observe that each accepting server session with a non-corrupted partner obtains, within the `ClientCertificateVerify` message, a valid signature output by an honest client on the (hashed) transcript $\texttt{ClientHello}\| \ldots \|\texttt{ClientCertificate}$, which exactly relates to that transcript as no hash collisions occurred. Together with the `ClientCertificateVerify` message itself, this makes the honest client agree on the transcript and all material entering the key derivation.[45] Hence, in particular, for each such server session that accepts there exists a client session that already accepted, holds the same session identifier, and derives the same session key.

- For $\mathsf{Pred} = \mathsf{AlmostConf}(\mathcal{C},\mathcal{S})$, note that the signature sent in `ServerCertificateVerify` is computed over (the non-colliding hash of) the transcript up to the `ServerCertificate` message which, along with the `ServerCertificateVerify` message itself (output by an honest server session), forms the key-confirmation identifier kcid. Therefore, for any accepting client session ℓ, there exists an honest server session ℓ' sharing the same kcid.

  Furthermore, no second honest client will send a `ClientCertificateVerify` message with a signature over a (hash of a) transcript containing the same kcid contents due to unique nonces. Also, no server obtains a forged signature and no hash collision occurs. Thus the server session ℓ' will only accept when receiving this client session's `ClientCertificateVerify` message. As the values signed in `ClientCertificateVerify`

[45] This argument, and likewise the arguments for the AlmostConf and KCIDbind predicates, make the strong unforgeability in the step to game $G^{\mathsf{Pred}'''}_{\texttt{draft-10-nf-m},\mathcal{A}}$ necessary. As the exchanged signatures themselves enter the key derivation (as well as session and confirmation identifiers), we need to ensure agreement on the actual signature (and hence rely on SUF-CMA security for this step) rather than only on the signed message (for which EUF-CMA suffices, cf. the MSKE analyses in Chapters 6 and 7). Otherwise, an adversary may be able to craft a different signature for the same (hashed-transcript) message, which, entering the key derivation, leads the two sessions to derive different keys.

together with `ClientCertificateVerify` uniquely determine the key derivation, session ℓ' will, if at all, accept with the same session key.

- For $\mathsf{Pred} = \mathsf{KCIDbind}$, the first condition that equal session identifiers imply equal key-confirmation identifiers follows immediately from defining kcid to be a prefix of sid.

  The second condition is satisfied for the same reasons that make sessions holding the same kcid in the $\mathsf{AlmostConf}$ predicate derive the same key. Again, kcid contains the transcript up to the `ServerCertificateVerify` message, which is signed by the client within its `ClientCertificateVerify` message that no second honest client will generate (and which can, at this point, neither be forged nor be derived due to colliding nonces or hashes). Therefore, if a client accepts (as the first of two sessions sharing the same kcid), it outputs the only `ClientCertificateVerify` message the server session will accept. As `ClientCertificateVerify` fixes the complete transcript which fully determines the session identifier and derived keys, both sessions will necessarily be partnered and derive the same key as required.

In other words, the predicate Pred (being $\mathsf{FullConf}(\mathcal{S},\mathcal{C})$, $\mathsf{AlmostConf}(\mathcal{C},\mathcal{S})$, resp. $\mathsf{KCIDbind}$) is always satisfied in $G^{\mathsf{Pred}'''}_{\texttt{draft-10-nf-m},\mathcal{A}}$ and thus

$$\mathsf{Adv}^{\mathsf{Pred}'''}_{\texttt{draft-10-nf-m},\mathcal{A}} = 0. \qquad \square$$

Theorem 8.9. *The TLS 1.3* `draft-10-nf-u` *full (EC)DHE handshake for unilateral authentication without finished messages satisfies full $(\mathcal{C},\mathcal{S})$-key confirmation. Formally, for any efficient adversary $\mathcal{A}$ against full $(\mathcal{C},\mathcal{S})$-key confirmation there exist efficient algorithms $\mathcal{B}_1$, $\mathcal{B}_2$ such that*

$$\mathsf{Adv}^{\mathsf{FullConf}(\mathcal{C},\mathcal{S})}_{\texttt{draft-10-nf-u},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_u \cdot \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2},$$

where n_u is the maximum number of users.

Proof. We show that, for unilateral authentication, the `ServerCertificateVerify` message obtained by a client session ensures that the sending server session has already accepted with the same session key and is partnered.

Again, we first modify the original $G^{\mathsf{FullConf}(\mathcal{C},\mathcal{S})}_{\texttt{draft-10-nf-u},\mathcal{A}}$ game in two steps similar to those in the proof of Theorem 8.8. First, we identically bound the probability that two honest sessions compute a colliding hash value for two different inputs by $\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}$ for an efficient reduction $\mathcal{B}_1$. We then ensure that no client session obtains a `ServerCertificateVerify` message containing a valid signature under an honest server's public key which was not generated by any honest session (recall that there is no `ClientCertificateVerify` message sent as the client does not authenticate). Similarly to the proof of Theorem 8.8 we can bound the probability that this happens by the advantage $\mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2}$ of an adversary $\mathcal{B}_2$ times a factor n_u for guessing the right identity, again using its signing oracle for the challenged server identity.

Due to these modifications, we are now assured of unconditional full $(\mathcal{C},\mathcal{S})$-key confirmation: the signature sent within `ServerCertificateVerify` together with that message itself fully determines the session identifier and key derivation and, hence, whenever a client accepts, the (honest) server session sending sending the `ServerCertificateVerify` message already accepted with the same session identifier and session key. $\square$

8.3.2 Key Confirmation with Finished Messages

Coming back to the original TLS 1.3 `draft-10` handshake, it is easy to see that the proof in Theorem 8.8 for key confirmation under mutual authentication in `draft-10-nf` immediately

carries over to the draft-10 handshake with mutual authentication (short: draft-10-m). For this, recall that the ServerFinished and ClientFinished messages do not enter the session identifiers or key derivation, which means they can essentially be treated as an "arbitrary bitstring" following the ServerCertificateVerify, resp. ClientCertificateVerify, message. Hence, we can apply the identical proof to the draft-10 mutually authenticating handshake to establish the same key confirmation properties.

Theorem 8.10. *The TLS 1.3* draft-10-m *full (EC)DHE handshake for mutual authentication with finished messages satisfies full $(\mathcal{S},\mathcal{C})$-key confirmation and almost-full $(\mathcal{C},\mathcal{S})$-key confirmation. Formally, for any efficient adversary $\mathcal{A}$ against full $(\mathcal{S},\mathcal{C})$-key confirmation, resp. almost-full $(\mathcal{C},\mathcal{S})$-key confirmation, there exist efficient algorithms $\mathcal{B}_1$, $\mathcal{B}_2$ such that*

$$\mathsf{Adv}^{\mathsf{Pred}}_{\texttt{draft-10-m},\mathcal{A}} \leq n_s^2 \cdot 2^{-|nonce|} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_u \cdot \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2},$$

for $\mathsf{Pred} = \mathsf{FullConf}(\mathcal{S},\mathcal{C})$*, resp.* $\mathsf{Pred} = \mathsf{AlmostConf}(\mathcal{C},\mathcal{S})$*, where n_u is the maximum number of users, n_s is the maximum number of sessions, and $|nonce| = 256$ is the bit-length of the nonces.*

Interestingly, when it comes to unilateral authentication, the ClientFinished messages sent as the single additional message from the client to the server changes the order in which the session key is accepted (the client accepting first here) and, hence, necessarily renders full $(\mathcal{C},\mathcal{S})$-key confirmation (as for draft-10-nf-u) unachievable. We can however show that clients indeed still enjoy almost-full $(\mathcal{C},\mathcal{S})$-key confirmation for the unilaterally authenticating draft-10 handshake (draft-10-u).

Theorem 8.11. *The TLS 1.3* draft-10-u *full (EC)DHE handshake for unilateral authentication with finished messages satisfies almost-full $(\mathcal{C},\mathcal{S})$-key confirmation. Formally, for any efficient adversary $\mathcal{A}$ against almost-full $(\mathcal{C},\mathcal{S})$-key confirmation there exist efficient algorithms $\mathcal{B}_1$, $\mathcal{B}_2$ such that*

$$\mathsf{Adv}^{\mathsf{AlmostConf}(\mathcal{C},\mathcal{S})}_{\texttt{draft-10-u},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_u \cdot \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2},$$

where n_u is the maximum number of users.

Proof. We show that the ServerCertificateVerify message obtained from an honest server ensures this server agrees on kcid and will, on acceptance, be partnered and derive the same key.

After excluding hash collisions and signature forgeries within ServerCertificateVerify again as in the proof of Theorem 8.9, the (unforged) ServerCertificateVerify message, computed over (the non-colliding hash of) the transcript, ensures the sending server agrees on the same key-confirmation identifier kcid.

Furthermore, the kcid value already uniquely determines the session identifier and derived key (this in particular renders requiring unique nonces unnecessary in this case): observe that $\mathsf{kcid} = \texttt{ClientHello}\| \ldots \|\texttt{ServerCertificateVerify}$ contains all messages affecting the session identifier and key derivation as (for unilateral authentication) messages ClientCertificate and ClientCertificateVerify are not sent and ServerFinished as well as ClientFinished are not included in the session hashes. Therefore, any two (accepting) sessions agreeing on the same kcid (and hence also sid) will derive the same session key and hence $\mathsf{AlmostConf}(\mathcal{C},\mathcal{S})$ as well KCIDbind are satisfied. □

To summarize, our analysis shows that key confirmation in the TLS 1.3 draft-10 full handshake can already be established through the exchanged CertificateVerify messages (given strong unforgeability of the deployed signature scheme) and, hence, not necessarily relies on the Finished messages included (also) for this purpose. This unveils a potential misconception that MACs over the transcript (in form of the Finished messages) are always necessary to achieve key confirmation.

Yet, our proofs should not be understood as an argument to remove finished messages from the TLS 1.3 handshake design, for several reasons. First of all are there further handshake modes in TLS 1.3 beyond the full (EC)DHE handshake, namely those based on pre-shared keys and for 0-RTT handshakes, cf. Chapters 6 and 7. These variants do not necessarily exchange `CertificateVerify` messages and, hence, rely on the `Finished` messages to provide both authentication as well as key confirmation.

Furthermore, our results for key confirmation, in contrast to the (multi-stage) key secrecy analyses in Chapters 6 and 7, rely on the strong (instead of existential) unforgeability of the deployed signature scheme. It may hence be preferable, to (in parallel also) rely on the `Finished` messages for key confirmation and, along with that, (existential) unforgeability of HMAC together with the same assumptions employed in the key secrecy analyses (e.g., on the Diffie–Hellman handshake or security of HKDF) necessary to argue security of the finished secrets.

Finally, later TLS 1.3 drafts include the option for 0.5-RTT data being sent by the server prior to the client's last flight. As seen in the analysis of the `draft-12` (EC)DHE 0-RTT handshake, deriving the application traffic key for that purpose from an abbreviated transcript makes (unforgeability of) the `Finished` messages already a requirement for the handshake's key secrecy.

Part II

Secure Channels

Chapter 9

Secure Channel Preliminaries

Summary. In this chapter we recap preliminaries on secure channels. We in particular summarize the established game-based formalization of a secure channel in terms of stateful authenticated encryption by Bellare, Kohno, and Namprempre [BKN02, BKN04]. On the way to introducing their notion, we will recall the definitions for classical symmetric encryption as well as for authenticated encryption with associated data introduced by Rogaway [Rog02].

9.1 Symmetric Encryption

As their most prominent goal, secure channels should protect the *confidentiality* of a communication between two parties. To this end, the two parties will regularly have run a key exchange protocol first and established a shared symmetric key. Naturally, the basic primitive for ensuring confidentiality in a secure channel is hence the classical concept of *symmetric encryption*.

A symmetric encryption scheme SE consists of three efficient algorithms. The probabilistic *key generation* algorithm KGen on input of a security parameter 1^λ outputs the shared symmetric key K (from some keyspace $\mathcal{K}$). Regularly, the key generation algorithm simply samples a key uniformly at random from the key space (i.e., $K \xleftarrow{\$} \mathcal{K}$), which we will assume from now on and hence always omit an explicit key generation algorithm. The probabilistic *encryption* algorithm Enc on input a key K and a message $m \in \{0,1\}^*$ outputs a ciphertext $c \in \{0,1\}^*$ (for simplicity, we here consider messages and ciphertexts to be arbitrary bit strings rather than restricting ourselves to specific message and ciphertext spaces). The deterministic *decryption* algorithm Dec on input a key K and a ciphertext c outputs a message m or possibly a distinguished error symbol $\perp \notin \{0,1\}^*$ to indicate a decryption failure. Correctness demands that, for any $K \xleftarrow{\$} \mathcal{K}$, any message $m \in \{0,1\}^*$, and any choice of randomness in the algorithms it holds that $\mathsf{Dec}_K(\mathsf{Enc}_K(m)) = m$, where we conveniently write the key input as subscript.

The basic security goal of confidentiality is captured via the notion of *indistinguishability under chosen-plaintext attacks* ($\mathsf{IND\text{-}CPA}$), going back to the seminal ideas of Goldwasser and Micali [GM84]. This notion, which we define more formally in Section 9.2.1 below for authenticated encryption, intuitively demands that an adversary cannot distinguish the encryption of any two messages of its own choice. A stronger variant, called *indistinguishability under chosen-ciphertext attacks* ($\mathsf{IND\text{-}CCA}$) and going back to the works by Naor and Yung [NY90] and Rackoff and Simon [RS92], demands encryptions of two messages being indistinguishability even when the adversary is given the power to decrypt (other) ciphertexts.

9.2 Authenticated Encryption (with Associated Data)

For a secure channel, we however not only care about the confidentiality of messages exchanged, but also their *integrity*: no intermediate party should be able to modify the communication between the two endpoint parties.

Bellare and Namprempre [BN00] coined the notion of *authenticated encryption* (AE) for a symmetric encryption scheme providing the combination of both confidentiality and integrity of plaintext messages. *Integrity of plaintexts* (INT-PTXT) here formalizes that an adversary is not able to forge a valid ciphertext that decrypts to a previously unseen message (i.e., a new message). Bellare and Rogaway [BR00] considered an even stronger variant, *integrity of ciphertexts* (INT-CTXT), which captures that an adversary is unable to forge an new valid ciphertext, even one decrypting to a previously seen message.

In real-world applications, it is often necessary that some parts of the exchanged communication is accessible to intermediate parties. For example, routing information for a message in a packet header of an Internet Protocol [Pos81a] packet must be accessible in transport. Still, such routing information should be integrity-protected against adversarial modification on the way. For such purposes, Rogaway [Rog02] established the notion of *authenticated encryption with associated data* (AEAD), which emerged as the core building block for secure channels (and other cryptographic components) to date. The notion of AEAD aims at jointly achieving the three described goals: providing confidentiality and integrity for the message encrypted as well as integrity for some additional string, the associated data *ad*. An AEAD scheme AEAD takes the associated data value $ad \in \{0,1\}^*$ as additional input in the Enc and Dec algorithms. In order to avoid the use of randomness (as a potential source of weaknesses) in the encryption process, the encryption algorithm can be made deterministic by taking a *nonce* $N \in \{0,1\}^n$ (a non-repeating number like a counter) as additional input which is also provided to the decryption algorithm; we then call the AEAD scheme *nonce-based*, without nonces we call it *randomized*. Putting everything together, correctness for a (nonce-based) AEAD scheme then demands that for any key $K \xleftarrow{\$} \mathcal{K}$, any message $m \in \{0,1\}^*$, any associated data $ad \in \{0,1\}^*$, and any nonce $N \in \{0,1\}^n$ it holds that $\mathsf{Dec}_K(N, ad, \mathsf{Enc}_K(N, ad, m)) = m$.

9.2.1 Stateless Notions for Confidentiality and Integrity

We can now formalize the security notions for (authenticated) symmetric encryption (with associated data) that capture confidentiality and integrity. We will do so in the syntax of nonce-based AEAD. Note that other notions can be easily obtained: for randomized AEAD by dropping the nonce input and making the encryption algorithm randomized, for authenticated encryption without associated data by additionally dropping the associated data input, and for classical symmetric encryption by additionally omitting the integrity requirements.

Confidentiality. Confidentiality captures the idea that an adversary is unable to distinguish the encryption of two messages of its choice, it comes in the variants of indistinguishability under chosen-plaintext attacks (IND-CPA) and under chosen-ciphertext attacks (IND-CCA). The security experiments for both notions (which we state formally in Figure 9.1) have in common that the adversary $\mathcal{A}$ is given access to a left-or-right encryption oracle $\mathcal{O}_{\mathsf{LoR}}$ which on input two messages m_0 and m_1 of equal length ($|m_0| = |m_1|$), along with a nonce N and associated data ad, always outputs the encryption $\mathsf{Enc}_K(N, ad, m_b)$, for a key $K \xleftarrow{\$} \mathcal{K}$ and a bit $b \xleftarrow{\$} \{0,1\}$ fixed in the experiment and unknown to $\mathcal{A}$. The adversary is then asked to distinguish between the cases $b = 0$ and $b = 1$, and if it cannot do so with non-negligible probability in the security parameter, the scheme is said to be IND-CPA-secure. For the stronger IND-CCA security notion, the adversary is additionally given access to a decryption oracle $\mathcal{O}_{\mathsf{Dec}}$ which it may query on

$\underline{\mathsf{Expt}^{\mathsf{IND\text{-}ATK},b}_{\mathsf{AEAD},\mathcal{A}}(1^\lambda)}$:
1 $K \xleftarrow{\$} \mathcal{K}$
2 $Q \leftarrow \emptyset$
3 $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}},[\mathcal{O}_{\mathsf{Dec}}]_{\mathsf{ATK=CCA}}}(1^\lambda)$
4 return b'

$\underline{\mathcal{O}_{\mathsf{LoR}}(N, ad, m_0, m_1)}$:
5 if $|m_0| \neq |m_1|$ then
6 return $\lightning$
7 $c \leftarrow \mathsf{Enc}_K(N, ad, m_b)$
8 $Q \leftarrow Q \cup \{(N, ad, c)\}$
9 return c

$\underline{\mathcal{O}_{\mathsf{Dec}}(N, ad, c)}$:
10 $m \leftarrow \mathsf{Dec}_K(N, ad, c)$
11 if $(N, ad, c) \notin Q$ then
12 return m
13 else
14 return $\lightning$

$\underline{\mathsf{Expt}^{\mathsf{INT\text{-}ATK}}_{\mathsf{AEAD},\mathcal{A}}(1^\lambda)}$:
1 $K \xleftarrow{\$} \mathcal{K}$
2 $Q \leftarrow \emptyset$
3 $\mathsf{win} \leftarrow 0$
4 $\mathcal{A}^{\mathcal{O}_{\mathsf{Enc}},\mathcal{O}_{\mathsf{Dec}}}(1^\lambda)$
5 return win

$\underline{\mathcal{O}_{\mathsf{Enc}}(N, ad, m)}$:
6 $c \leftarrow \mathsf{Enc}_K(N, ad, m)$
7 if $\mathsf{ATK} = \mathsf{PTXT}$ then
8 $Q \leftarrow Q \cup \{(N, ad, m)\}$
9 else // $\mathsf{ATK} = \mathsf{CTXT}$
10 $Q \leftarrow Q \cup \{(N, ad, c)\}$
11 return c

$\underline{\mathcal{O}_{\mathsf{Dec}}(N, ad, c)}$:
12 $m \leftarrow \mathsf{Dec}_K(N, ad, c)$
13 if $m \neq \bot$ then
14 if $\mathsf{ATK} = \mathsf{PTXT}$ and $(N, ad, m) \notin Q$ then
15 $\mathsf{win} \leftarrow 1$
16 if $\mathsf{ATK} = \mathsf{CTXT}$ and $(N, ad, c) \notin Q$ then
17 $\mathsf{win} \leftarrow 1$
18 return m

Figure 9.1: Security experiments for *confidentiality* (IND-ATK) and *integrity* (INT-ATK) of AEAD schemes, where ATK is a placeholder for CPA or CCA, resp. PTXT or CTXT. The brackets $[\mathcal{O}_{\mathsf{Dec}}]_{\mathsf{ATK=CCA}}$ indicate that only the IND-CCA adversary has access to the $\mathcal{O}_{\mathsf{Dec}}$ oracle.

ciphertexts (along with a nonce and associated data) distinct from those obtained from the $\mathcal{O}_{\mathsf{LoR}}$ oracle, i.e., on any distinct combination of ciphertext, nonce, and associated data (and hence especially on newly crafted ciphertexts). The latter restriction is to prevent a trivial win of the adversary by querying $\mathcal{O}_{\mathsf{LoR}}$ on some distinct m_0 and m_1 along with some nonce N and associated data ad to obtain a ciphertext c, then query $\mathcal{O}_{\mathsf{Dec}}$ on N, ad, and c to obtain a decryption m, and output 0 in case $m = m_0$ and 1 otherwise.

Integrity. Integrity captures the idea that an adversary cannot tamper with the messages respectively ciphertexts exchanged and accordingly comes in the variant of plaintext and ciphertext integrity (INT-PTXT, resp. INT-CTXT). In the security experiments (see Figure 9.1), the adversary is given encryption and decryption oracles $\mathcal{O}_{\mathsf{Enc}}$ and $\mathcal{O}_{\mathsf{Dec}}$, which compute $c \leftarrow \mathsf{Enc}_K(N, ad, m)$, resp. $m \leftarrow \mathsf{Dec}_K(N, ad, c)$, for inputs of the adversary's choice. The adversary is declared successful (via a flag $\mathsf{win} = 1$, initialized to 0) if it submits to its decryption oracle a tuple of nonce N, associated data ad, and ciphertext c that decrypts to a non-error message $m \neq \bot$, given that (N, ad, m) were never queried to $\mathcal{O}_{\mathsf{Enc}}$ (for INT-PTXT) resp. that c was never output on an $\mathcal{O}_{\mathsf{Enc}}$ query with nonce N and associated data ad (for INT-CTXT). That is, for INT-PTXT the adversary is required to forge an (N, ad, c) tuple resulting in a previously unseen message m, while for INT-CTXT any (N, ad, c) distinct from previous $\mathcal{O}_{\mathsf{Enc}}$ input/outputs is considered a valid forgery, even if it decrypts to a message m previously queried to $\mathcal{O}_{\mathsf{Enc}}$. Note that INT-CTXT security implies INT-PTXT security, as, conversely, a successful INT-PTXT forgery c such that $(N, ad, m) \notin Q$ in particular implies that for c also $(N, ad, c) \notin Q$ in the INT-CTXT experiment, as by correctness no other message m' together with N and ad could have been encrypted to c.

We can now formalize the different confidentiality and integrity security notions for AEAD schemes. It is easy to see that IND-CCA security implies IND-CPA security (by omitting the $\mathcal{O}_{\mathsf{Dec}}$ oracle). Furthermore, INT-CTXT security implies INT-PTXT security, as a distinct message decrypted in particular necessitates a distinct ciphertext input in the $\mathcal{O}_{\mathsf{Dec}}$ oracle.

Definition 9.1 (Security for AEAD schemes)**.** *Let* $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ *be a nonce-based AEAD scheme with key space* $\mathcal{K}$*.*

Let experiment $\mathsf{Expt}^{\mathsf{IND\text{-}ATK},b}_{\mathsf{AEAD},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *and a bit* $b \in \{0,1\}$ *be defined as in Figure 9.1, where* ATK *is a placeholder for either* CPA *or* CCA*. We say that* AEAD *provides* indistinguishability under chosen-plaintext attacks*, respectively,* chosen-ciphertext attacks *(*IND-CPA*, resp.* IND-CCA*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}ATK}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}ATK},1}_{\mathsf{AEAD},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}ATK},0}_{\mathsf{AEAD},\mathcal{A}}(1^\lambda) = 1\right]\right|.$$

Likewise, let experiment $\mathsf{Expt}^{\mathsf{INT\text{-}ATK},b}_{\mathsf{AEAD},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 9.1, where* ATK *is a placeholder for either* PTXT *or* CTXT*. We say that* AEAD *provides* integrity of plaintexts*, respectively,* ciphertexts *(*INT-PTXT*, resp.* INT-CTXT*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{INT\text{-}ATK}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{INT\text{-}ATK}}_{\mathsf{AEAD},\mathcal{A}}(1^\lambda) = 1\right].$$

9.3 Stateful Authenticated Encryption

Beyond confidentiality and integrity of messages, secure channels are supposed to also protect against replay, reordering, and dropping of messages within the sequence of messages sent in a communication. In order to be able to detect such modifications, the encryption and decryption algorithms need to be stateful. Bellare, Kohno, and Namprempre [BKN02, BKN04] introduced this notion formalized as *stateful authenticated encryption* in their work analyzing, for the first time, a real-world channel protocol (i.e., the Secure Shell (SSH) protocol [YL06a]), establishing the by now widely accepted game-based cryptographic security model for channels.

Formally, in the stateful AE setting the key generation algorithm is replaced with a more generic (probabilistic) *initialization* algorithm Init, outputting (beyond the shared key) initial sending and receiving states st_S, resp. st_R; we write $(K, \mathsf{st}_S, \mathsf{st}_R) \xleftarrow{\$} \mathsf{Init}(1^\lambda)$. Encryption and decryption then obtain and (independently) update the respective state information, i.e., $(\mathsf{st}'_S, c) \xleftarrow{\$} \mathsf{Enc}_K(\mathsf{st}_S, m)$ and $(\mathsf{st}'_R, m) \xleftarrow{\$} \mathsf{Dec}_K(\mathsf{st}_R, c)$. Correctness for stateful authenticated encryption then requires that, starting from any initial output of Init, when Enc is invoked on a sequence of messages $m_1, m_2, \ldots, m_n$ (with updated states) resulting in ciphertexts $c_1, c_2, \ldots, c_n$, processing these ciphertexts in the same order (with updated states) by Dec yields the original message sequence $m_1, m_2, \ldots, m_n$ again.

As for stateless authenticated encryption, one in principle can readily augment the syntax of stateful AE with associated data (nonces in contrast would usually be subsumed by the updated state). From a structural perspective, associated data however belongs rather to the network layer of a communication (protecting, e.g., routing or other administrative information) and AEAD is hence useful as a core building block for secure channels (as we will also see in Chapters 10–12). Thinking of stateful authenticated encryption as providing a secure-channel interface to the application layer in the network stack, applications in contrast expect to transmit messages only, without distinction of associated data. We hence choose not to add an associated-data component in our formalization of stateful authenticated encryption.

9.3.1 Stateful Notions for Confidentiality and Integrity

The stateful variants of confidentiality and integrity notions for (authenticated) symmetric encryption now need to additionally capture the security guarantees against replay, reordering, and dropping of messages. Moreover, an adversary needs to be able to simulate honest communication (including updating states) up to a certain point at which it launches its attack on the scheme, e.g., by interfering with the communication. On a high level, this is done

$\underline{\mathsf{Expt}^{\mathsf{IND\text{-}sfATK},b}_{\mathsf{sfAE},\mathcal{A}}(1^\lambda)}$:
1 $(K, \mathsf{st}_S, \mathsf{st}_R) \xleftarrow{\$} \mathsf{Init}(1^\lambda)$
2 $i, j \leftarrow 0$
3 $\mathsf{sync} \leftarrow 1$
4 $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}}, [\mathcal{O}_{\mathsf{Dec}}]_{\mathsf{ATK=CCA}}}(1^\lambda)$
5 return b'

$\underline{\mathcal{O}_{\mathsf{LoR}}(m_0, m_1)}$:
6 if $|m_0| \neq |m_1|$ then
7 return $\lightning$
8 $i \leftarrow i + 1$
9 $(\mathsf{st}_S, c) \xleftarrow{\$} \mathsf{Enc}_K(\mathsf{st}_S, m_b)$
10 $c_i \leftarrow c$
11 return c

$\underline{\mathcal{O}_{\mathsf{Dec}}(c)}$:
12 $j \leftarrow j + 1$
13 $(\mathsf{st}_R, m) \leftarrow \mathsf{Dec}_K(\mathsf{st}_R, c)$
14 if $j > i$ or $c \neq c_j$ then
15 $\mathsf{sync} \leftarrow 0$
16 if $\mathsf{sync} = 0$ then
17 return m
18 else
19 return $\lightning$

$\underline{\mathsf{Expt}^{\mathsf{INT\text{-}sfATK}}_{\mathsf{sfAE},\mathcal{A}}(1^\lambda)}$:
1 $(K, \mathsf{st}_S, \mathsf{st}_R) \xleftarrow{\$} \mathsf{Init}(1^\lambda)$
2 $i, j \leftarrow 0$
3 $\mathsf{sync} \leftarrow 1$
4 $\mathsf{win} \leftarrow 0$
5 $\mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Dec}}}(1^\lambda)$
6 return win

$\underline{\mathcal{O}_{\mathsf{Enc}}(m)}$:
7 $i \leftarrow i + 1$
8 $(\mathsf{st}_S, c) \xleftarrow{\$} \mathsf{Enc}_K(\mathsf{st}_S, m)$
9 $c_i \leftarrow c$
10 $m_i \leftarrow m$
11 return c

$\underline{\mathcal{O}_{\mathsf{Dec}}(c)}$:
12 $j \leftarrow j + 1$
13 $(\mathsf{st}_R, m) \leftarrow \mathsf{Dec}_K(\mathsf{st}_R, c)$
14 if $\mathsf{ATK} = \mathsf{PTXT}$ then
15 if $m \neq \perp$ and $(j > i$ or $m \neq m_j)$ then
16 $\mathsf{win} \leftarrow 1$
17 if $\mathsf{ATK} = \mathsf{CTXT}$ then
18 if $j > i$ or $c \neq c_j$ then
19 $\mathsf{sync} \leftarrow 0$
20 if $\mathsf{sync} = 0$ and $m \neq \perp$ then
21 $\mathsf{win} \leftarrow 1$
22 return m

Figure 9.2: Security experiments for *confidentiality* (IND-sfATK) and *integrity* (INT-sfATK) of stateful authenticated encryption schemes, where ATK is a placeholder for CPA or CCA, resp. PTXT or CTXT. The brackets $[\mathcal{O}_{\mathsf{Dec}}]_{\mathsf{ATK=CCA}}$ indicate that only the IND-sfCCA adversary has access to the $\mathcal{O}_{\mathsf{Dec}}$ oracle.

by the experiments keeping a sequence of the ciphertexts sent using the encryption oracle, and consider an adversary passive as long as it simply relays the sent ciphertexts in correct order to the decryption oracle. As soon as it deviates from the sequence of sent ciphertexts, it is considered active in the sense of making the encryption and decryption processes (and states) losing synchronization (we say: they go "out of sync"), and from that point on the experiment considers attacks, e.g., in forging ciphertexts.

Confidentiality. Confidentiality is again modeled as indistinguishability of the outputs a left-or-right encryption oracle, for the CCA variant again given a decryption oracle $\mathcal{O}_{\mathsf{Dec}}$, see the formal experiment definitions in Figure 9.2. Beyond the obvious syntactical changes, the resulting IND-sfCPA is essentially the same as the IND-CPA one, the CCA decryption oracle however needs a substantial modification in order to reflect (protection against) replays, reordering, and dropping of messages.

In the stateless setting, the decryption oracle would reject on any previously ciphertext output by the $\mathcal{O}_{\mathsf{LoR}}$ encryption oracle (independent of the ordering) to prevent trivial attacks. Now, we want to suppress the output of decryption only as long as the ciphertext sequence received (including the queried one) exactly match the sequence of ciphertexts produced by $\mathcal{O}_{\mathsf{LoR}}$; otherwise the adversary is given the decryption output. This captures the following intuition: as long as the adversary relays ciphertexts in their original ordering, decryption will yield the (challenge) messages input to $\mathcal{O}_{\mathsf{LoR}}$ by correctness, and hence its output must be suppressed to avoid trivial wins. However, as soon as the adversary deviates from the original ciphertext sequence (indicated by the synchronization flag sync being set to 0), the decryption algorithm should be able to detect this and its output should not yield any information on the original messages, which is modeled by providing this output to the adversary. Note that deviation is

persistent: once the adversary deviates from the original sequence, all follow-up queries—even relaying original ciphertexts again—will be considered "out of sync" as well and output be provided to the adversary.

Integrity. Integrity as before comes in two flavors, where INT-sfPTXT straightforwardly calls an adversary successful if it manages to feed a sequence of ciphertexts into the decryption oracle such that the resulting sequence of messages deviates from the one encrypted using the $\mathcal{O}_{\mathsf{Enc}}$ oracle. Ciphertext integrity again formalizes a stronger demand, namely that it should be impossible for an adversary to come up with a deviating sequence of ciphertexts leading to valid (i.e., non-error) outputs on the decryption side. For the latter notion, the same concept of (losing) synchronization is applied as for IND-sfCCA security: decrypted messages are considered (i.e., checked for being non-errors) from the point on where synchronization is lost ($\mathsf{sync} = 0$) in terms of the adversary deviating from the original ciphertext series produced by $\mathcal{O}_{\mathsf{Enc}}$. Both notions hence in particular encode protection against replay, reordering, or dropping of messages, resp. ciphertexts, as an adversary wins if a stateful AE scheme does not detect such attacks by outputting error messages $\perp$.

We can now also formalize the different notions of confidentiality and integrity for stateful AE schemes. Their relation is as for the stateless setting: Again, the implication IND-sfCCA $\implies$ IND-sfCPA holds directly by omitting the $\mathcal{O}_{\mathsf{Dec}}$ oracle. Also, a (non-error) deviation in the message sequence decrypted in $\mathcal{O}_{\mathsf{Dec}}$ requires a deviation in the ciphertext and hence INT-sfCTXT implies INT-sfPTXT.

Definition 9.2 (Security for stateful AE schemes)**.** *Let* $\mathsf{sfAE} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *be a stateful AE scheme.*

Let experiment $\mathsf{Expt}^{\mathsf{IND\text{-}sfATK},b}_{\mathsf{sfAE},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *and a bit* $b \in \{0,1\}$ *be defined as in Figure 9.2, where* ATK *is a placeholder for either* CPA *or* CCA*. We say that* sfAE *provides* stateful indistinguishability under chosen-plaintext attacks*, respectively,* chosen-ciphertext attacks *(*IND-sfCPA*, resp.* IND-sfCCA*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}sfATK}}_{\mathsf{sfAE},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}sfATK},1}_{\mathsf{sfAE},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}sfATK},0}_{\mathsf{sfAE},\mathcal{A}}(1^\lambda) = 1\right]\right|.$$

Likewise, let experiment $\mathsf{Expt}^{\mathsf{INT\text{-}sfATK},b}_{\mathsf{AEAD},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 9.2, where* ATK *is a placeholder for either* PTXT *or* CTXT*. We say that* sfAE *provides* stateful integrity of plaintexts*, respectively,* ciphertexts *(*INT-sfPTXT*, resp.* INT-sfCTXT*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{INT\text{-}sfATK}}_{\mathsf{sfAE},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{INT\text{-}sfATK}}_{\mathsf{sfAE},\mathcal{A}}(1^\lambda) = 1\right].$$

9.4 Notation and Terminology

We conclude the preliminaries on channels by introducing some notation and terminology for our work on channels in the following chapters.

Notation. Let Σ be an alphabet and $s \in \Sigma^*$ be a string, where $s = \varepsilon$ denotes the empty string. By $|s|$ we denote the length of the string s, by $s[i] \in \Sigma$ the i-th character of the string (where $s[1]$ is the first character for a non-empty string), and by $s[i, \ldots, j]$ the substring from and including $s[i]$ up to and including $s[j]$, i.e., $s[i, \ldots, j] = s[i] \| \ldots \| s[j]$. Given two strings $s, t \in \Sigma^*$ we write $s \preccurlyeq t$ to indicate that s is a *prefix* of t, i.e., there exists $r \in \Sigma^*$ such that $s \| r = t$; in

this case we write $r = t \,\%\, s$. Similarly, we write $s \prec t$ to indicate that s is a *strict prefix* of t, i.e., $s \preccurlyeq t$ and $s \neq t$. We denote the longest common prefix of s and t by $[s,t] = [t,s]$. Note that $s \preccurlyeq t$ is equivalent to having $[s,t] = s$. With the above notation $s \,\%\, [s,t]$ denotes the suffix of s with the longest common prefix of s and t stripped off.

Let $\mathbf{s} = (s_1, \ldots, s_\ell) \in (\Sigma^*)^\ell$ be a vector of strings for some integer ℓ. (The strings need not be of equal length.) For all $0 \leq i \leq j \leq \ell$ we denote $\mathbf{s}[i] = s_i$ and $\mathbf{s}[i, \ldots, j] = (s_i, \ldots, s_j)$. We say that two vectors $\mathbf{s} = (s_1, \ldots, s_\ell) \in (\Sigma^*)^\ell$ and $\mathbf{t} = (t_1, \ldots, t_{\ell'}) \in (\Sigma^*)^{\ell'}$ are equal, and write $\mathbf{s} = \mathbf{t}$, if and only if $\ell = \ell'$ and for all $i \in \{1, \ldots, \ell\}$ it holds $s[i] = t[i]$. The conversion of a vector into a string is simply performed via the concatenation operation $\|\mathbf{s} = s_1 \| \ldots \| s_\ell$. By convention, the concatenation of an empty vector () is the empty string ε. Slightly overloading notation, we denote the merge of two vectors $\mathbf{s} = (s_1, \ldots, s_\ell)$ and $\mathbf{t} = (t_1, \ldots, t_{\ell'})$ as $\mathbf{s} \| \mathbf{t} = (s_1, \ldots, s_\ell, t_1, \ldots, t_{\ell'})$. We also indicate by $\mathbf{s} \preccurlyeq \mathbf{t}$ that $\mathbf{s}$ is a prefix of $\mathbf{t}$, i.e., $\ell \leq \ell'$ and $s_1 = t_1, \ldots, s_\ell = t_\ell$, and in this case we denote by $\mathbf{t} \,\%\, \mathbf{s}$ the (potentially empty) vector $\mathbf{u}$ such that $\mathbf{t} = \mathbf{s} \| \mathbf{u}$. We write $\mathbf{s} \prec \mathbf{t}$ to indicate that $\mathbf{s}$ is a strict prefix of $\mathbf{t}$. Similarly, we denote by $[\mathbf{s}, \mathbf{t}]$ the longest vector that is a prefix of both $\mathbf{s}$ and $\mathbf{t}$.

Channel Terminology. The terminology we use in Chapters 10–12 reflects the generic functionality that a channel should provide, i.e., allowing a sender to transmit messages and a receiver to obtain them in a reliable way. In particular, the notion of a channel is independent of the targeted security properties (like confidentiality or integrity) as these may vary from one specific application to another. While, e.g., a secure channel may generically be thought of as being realized via stateful authenticated encryption, an authenticated channel might choose to leave confidentiality aside and provide only integrity. We prefer to keep a higher level of abstraction and explicitly separate the generic notion of a channel from its building blocks or targeted security guarantees and hence define sending (Send) and receiving (Recv) algorithms rather than encryption and decryption algorithms.

Chapter 10

Stream-Based Channels

Summary. In this chapter we present our notion of stream-based channels, capturing the fragmentation of sent and received message streams in real-world channel protocols. We begin by defining the functional specification of a stream-based channel. We then lift the classical (stateful) notions for confidentiality and integrity of channels to the streaming setting. This turns out to result in a significant increase in complexity of the definitions due to the inherently more complex (and less structured) setting of message streams compared to the setting with distinct, atomic messages. We finally provide an AEAD-based construction of a stream-based channel achieving our strongest security notion. Being comparatively close to the TLS channel this construction additionally provides some validation of that protocol's design. The results in this chapter are based on a work published at CRYPTO 2015 [FGMP15] and its extended full version [FGMP17].

10.1 Introduction

As discussed in the previous chapter, authenticated encryption with associated data (AEAD) [Rog02] has emerged as being the right cryptographic tool for building secure channels. AEAD provides both confidentiality and integrity guarantees for data and can additionally protect the integrity of associated information. However, on its own, AEAD does not constitute a secure channel. For example, in most practical situations, a secure channel should provide more than simple encryption of messages, but also guarantee detection of (and possibly recovery from) out-of-order delivery and replays of messages. Furthermore, a secure channel should deal with error handling, with errors potentially arising from both cryptographic and non-cryptographic processing—whether or not to tear-down a secure channel session if an error is encountered, and how (and indeed whether) to signal errors to the other side. As another difference, some secure channel designs such as IPsec [KS05] (which provides security at the IP layer) and to a limited extent TLS [DR08] (providing application-layer security) have additional features that can be used to provide protection against traffic analysis. A secure channel may accept messages of arbitrary length and need to fragment these before encryption, and may reassemble these fragments again after decryption; alternatively, it may present to applications a maximum message size that is well-matched to the underlying network infrastructure. Finally, and most importantly in the context of this and the following chapter, a secure channel may be designed to protect a *stream* of data rather than the series of discrete messages that is usually found in cryptographic abstractions.

There is, then, a substantial gap between what the AEAD primitive can reasonably provide and the needs of secure channels. We are not the first to recognize this gap, of course. As already introduced in the previous chapter, Bellare, Kohno, and Namprempre [BKN02, BKN04]

extended the standard security notions of confidentiality and integrity for symmetric encryption to the stateful setting, enabling the treatment of security of the ordering of discrete messages in a secure channel, with application to the analysis of SSH being their principle motivation. We refer to Section 1.3 for a summary of the follow-up work on channels, e.g., introducing formalisms in different settings [Nam02, Sho99, Can00, MT10, BMM+15], capturing different levels in a security hierarchy [KPB03, BHMS16], or particular properties of specific protocols like TLS [PRS11, JKSS12, KPW13].

Data is a stream. Characteristic of all the above-mentioned prior works is that they treat secure channels as providing an *atomic* interface for messages, meaning that the channel is designed only for sending and receiving sequences of discrete messages. However, this only captures a fraction of secure channel designs that are actually used in the real world. In particular, TLS [DR08, Res18], SSH [YL06a], and QUIC [QUI] all provide a *streaming* interface for the applications that use them: applications submit fragments (or segments) of message (or plaintext) streams to an application programming interface (API), and similarly receive fragments of message streams from the API. The sending side may arbitrarily buffer and/or fragment the message stream before encapsulating it for sending.

Moreover, in some cases, even under normal operations, it is not guaranteed by the network that the resulting stream of ciphertext fragments (which we refer to as *ciphertexts* henceforth treating them as opaque bit strings) that is sent will arrive at the receiver with the same pattern of fragmentation, even if the reconstructed message streams are in the end identical.[46] Under adversarial conditions, such guarantees certainly do not hold: for example, TLS runs over TCP [Pos81b] and an active man-in-the-middle adversary can tinker with the TCP segments, adding, removing and reordering TLS data at will. Thus, practical secure channels need to securely process arbitrarily fragmented ciphertexts. Finally, to make things even more complex, and coming full circle, applications (like HTTP [FGM+97]) often rely on stream-oriented secure channels (like TLS) to securely deliver what are actually, in their semantics, atomic messages.

This discussion points to a mismatch between atomic descriptions of secure channels in the cryptography literature and the reality of the operation of secure channels. As one may expect, such mismatches can have negative consequences for security. The starkest example of this comes from the plaintext recovery attack against SSH given by Albrecht et al. [APW09]. Their attack specifically exploits the adversary's ability to deliver arbitrary sequences of SSH packet fragments to the receiver (over TCP) and observe the receiver's behavior in response. The attack is possible despite the analysis of Bellare et al. [BKN04] which proved that the SSH secure channel satisfies suitable *atomic* stateful security notions. Related attacks against certain IPsec configurations (and exploiting IPsec's need to handle IP fragmentation) were presented by Degabriele and Paterson [DP10]. Attacks highlighting a disjunction between what applications expect and what secure channels provide, in the specific context of HTTP and TLS, can be found in [SP13, BDF+14]. All these attacks highlight deficits of previous approaches to modeling and analyzing secure channels.

Boldyreva et al. [BDPS12] extended the classical, atomic secure-channel confidentiality notion to cover the case of SSH-like fragmentation in secure channels, broadening the SSH-specific work of Paterson and Watson [PW10]. Subsequently, and concurrently to our work [FGMP15, FGMP17], Albrecht et al. [ADHP16] augmented the model of [BDPS12] with a corresponding notion of integrity. However, while their work allows for fragmented delivery of ciphertexts to the receiver, it still assumes that the encryption process on the sender's side is atomic, meaning

[46] IPsec is a prime example; because of the interaction between IPsec and IP with its fragmentation features, IPsec-protected packets can arrive at their destination in a sequence of fragments, each fragment contained in its own IP packet, and possibly arriving out of order.

that there is a one-to-one correspondence between messages and ciphertexts. This may be the case for SSH when used in interactive sessions, but it is not the case for the tunneling mode of SSH, and never the case for other secure channel protocols. For example, even though the TLS specification [DR08, Res18] does not include a formal API definition, it is clear that the design intention is to provide a secure channel for data streams and the application programmer is in practice offered a TCP-like socket interface. As noted above, the sending side can arbitrarily buffer and fragment the message stream when preparing ciphertexts for sending.

Stream-based channels. In this chapter we develop formal functional specifications, security notions, and a construction (using AEAD as a building block) for *stream-based channels*. In Chapter 11 we will then explore how applications, given access to a stream-based channel meeting our security notions, can safely use it to transport atomic messages over a fragmenting network via an *atomic-message channel (supporting fragmentation)*.

Our models are in the game-based tradition, and extend those of [BKN04, BDPS12] to handle the streaming nature of the channels that we consider. Figure 10.1 gives an overview of the notions we establish for both stream-based and atomic-message channels and their relations, including formal and conceptual relations to the established notions of AEAD [Rog02] and symmetric encryption supporting fragmentation [BDPS12, ADHP16].

While our methodology and modeling closely resemble those of Boldyreva et al. [BDPS12], and indeed build upon them, a crucial difference comes in our treatment of the sending (or encrypting) function of a stream-based channel: in [BDPS12], this is still atomic (while decryption is not), whereas in our stream-based channel setting, both the sending and receiving function support streams of data, with potentially arbitrary buffering and fragmentation on the sending and receiving side. This requires careful modification of the confidentiality definitions of [BDPS12]. In addition, we develop suitable integrity notions for the streaming setting whereas [BDPS12] does not consider this aspect. This is important because the (informal) security properties that applications expect a secure channel to provide confidentiality as well as integrity. Concurrent to our work [FGMP15, FGMP17], Albrecht et al. [ADHP16] augmented the model of [BDPS12] with an integrity notion for an analysis of the SSH protocol.

Bringing integrity into the picture for stream-based channels also enables us to prove a composition result analogous to the classical result of [BN00] for symmetric encryption schemes, which states that IND-CPA security in combination with integrity of ciphertexts (INT-CTXT security) guarantees IND-CCA security. This provides an easy route to proving that a given stream-based channel construction provides the expected confidentiality (indistinguishability under chosen ciphertext-fragment attacks, or IND-CCFA security) and integrity (integrity of plaintext streams, INT-PST security).

The composition theorem brings an interesting technical challenge to surmount: as was already recognized by Boldyreva et al. [BDPS14] for the classical (atomic) setting, the possibility that realistic models of encryption schemes may involve *multiple* error messages means that the original composition proof of [BN00] does not go through. In [BDPS14], this was overcome by assuming the scheme is such that only one of the possible error messages has a non-negligible chance of being produced during operation of the scheme. Here we take a different tack, introducing the concept of *error predictability*, which guarantees the existence of an efficient algorithm that can predict which errors should be output during decryption of a ciphertext stream. We exemplify that such a predictor can exist for a given scheme, even in cases where the analogous conditions to those in [BDPS14] are not satisfied. Our approach can also be used in the atomic-message setting to extend the composition theorem to schemes with distinguishable but predictable errors.

We demonstrate the feasibility of our security notions by providing a generic construction for a

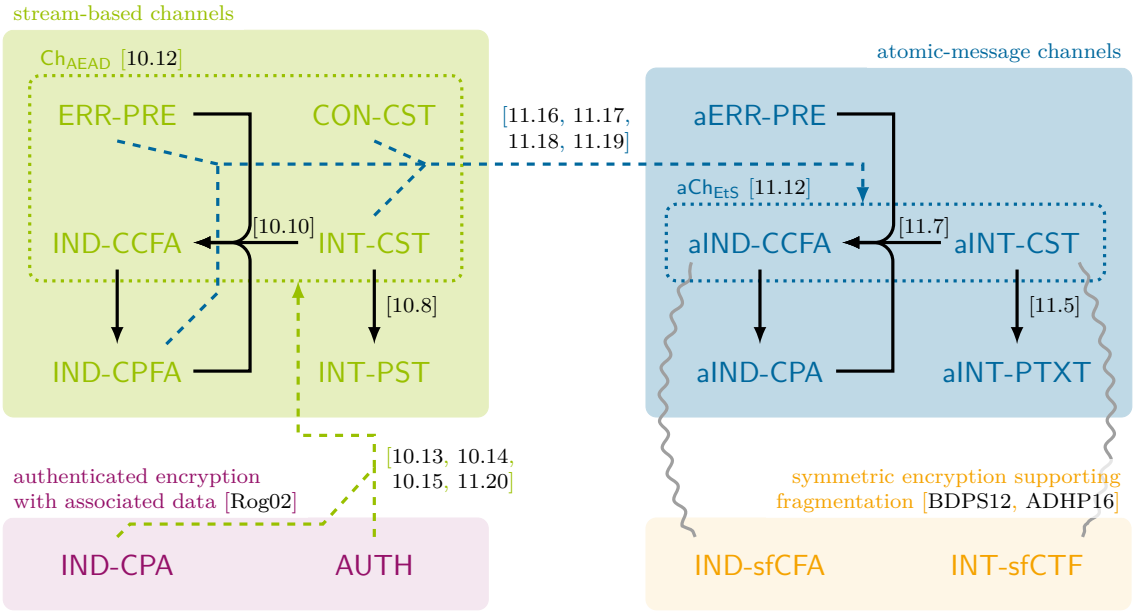


Figure 10.1: Overview of the notions we establish for stream-based channels (in Chapter 10) and atomic-message channels supporting fragmentation (in Chapter 11), as well as their formal or conceptual relations to established notions that we demonstrate for authenticated encryption with associated data (AEAD) [Rog02] and symmetric encryption supporting fragmentation [BDPS12, ADHP16]. We provide confidentiality notions in terms of (atomic-message) indistinguishability ((a)IND) under chosen plaintext, plaintext fragment and ciphertext fragment attacks (CPA/CPFA/CCFA), integrity notions in terms of (atomic-message) integrity ((a)INT) of plaintexts, plaintext streams and ciphertext streams (PTXT/PST/CST), and more specific notions for (atomic-message) error predictability ((a)ERR-PRE) and conciseness of ciphertext streams (CON-CST).
Filled rounded rectangular areas indicate the three conceptual notions for channels and the notion of AEAD. Solid arrows indicate implications we establish between security notions within these. Dotted rounded rectangles encompass the security notions achieved by our generic constructions $\mathsf{Ch}_{\mathsf{AEAD}}$ and $\mathsf{aCh}_{\mathsf{EtS}}$. Dashed arrows indicate the security notions required in these constructions from the corresponding building blocks. Wavy lines indicate conceptually analogous security notions between our setting of atomic-message channels supporting fragmentation and the notion of symmetric encryption supporting fragmentation [BDPS12, ADHP16]. Numbers in brackets refer to the corresponding theorems and constructions in Chapters 10 and 11. Further discussions can be found in the text.

stream-based channel that uses AEAD as a component and achieves our strongest confidentiality and integrity notions. The resulting stream-based channel closely mimics the TLS record protocol. That way, our security results provide validation for this important real-world protocol design, whilst fully taking its streaming behavior into account.

In Chapter 11, we will then take a look at applications that wish to send atomic messages over a channel protocol providing a streaming interface, and leverage our model for stream-based channel to establish conditions and constructions which enable security in such settings.

10.2 Syntax and Functionality of Stream-Based Channels

We capture the functionality of channel protocols that offer a reliable transmission of *streams* like the Transmission Control Protocol (TCP) [Pos81b] and, in a second step, we define confidentiality and integrity properties expected from (stream-based) secure channel protocols like the Transport Layer Security (TLS) record protocol [DR08] or the Secure Shell (SSH) Binary

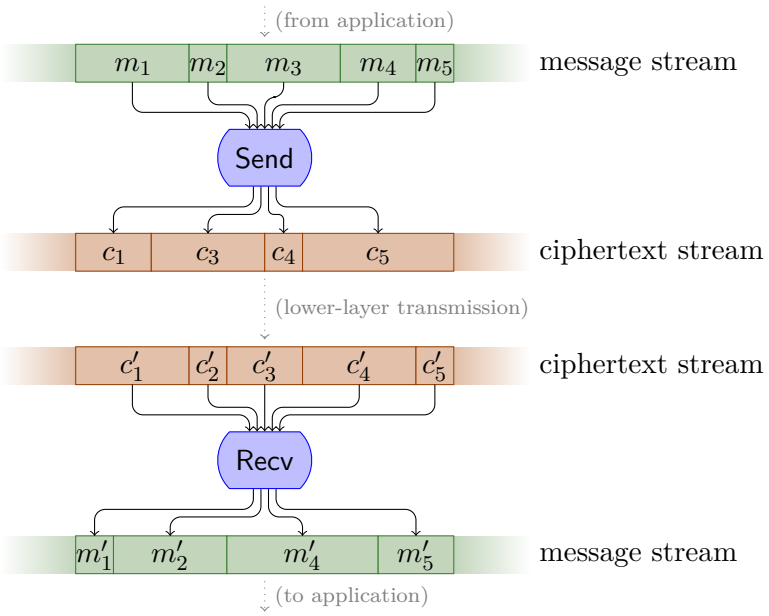


Figure 10.2: Illustration of the behavior of the Send and Recv algorithms of a stream-based channel, indicating the message and ciphertext fragments being sent (m_i resp. c_i) and received (m'_i resp. c'_i). Note that, due to buffering, output ciphertexts and messages (c_i resp. m'_i) can be empty.

Packet Protocol [YL06b].[47] To do so we first need to define the syntax of stream-based channels that, in contrast to previous models for channels, send fragments of a message (or plaintext) *stream* rather than atomic messages. In order to remain close to real-world implementations we restrict both the message space and the ciphertext space to the set of bit strings, where we understand 'messages' and 'ciphertexts' not as atomic units, but as fragments (i.e., substrings) of a message stream and a ciphertext stream.

Additionally, we do not stipulate a particular input/output behavior on the sender side, but instead allow the sending algorithm Send to process input data at its discretion, e.g., implementing some form of buffering before sending. We enforce sending out particular chunks of the message stream by employing the established concept of 'flushing a stream' known from network socket programming, and provide the Send algorithm with an additional *flush* flag $f \in \{0, 1\}$ which, if set to $f = 1$, ensures that all the message fragments fed so far are sent out instantaneously. Jumping ahead, in our security model this choice conservatively also allows the adversary to control the flush flag. If the flush flag is set to zero, Send may internally decide to keep accepting more message fragments or to send out a ciphertext fragment, depending on its implementation and resources.

We remark that our model also captures real-world channels that, instead of offering an explicit flushing mechanism, buffer their input until a specified timeout is reached. In such scenarios, the adversary is simply given control over the timeout through controlling the flush

[47] Our model inherently assumes that, in a benign scenario, ciphertext fragments are delivered reliably and in order (i.e., in a TCP-like manner). While we recognize that efficient and secure transmission protocols can be designed also on top of unreliable protocols like the User Datagram Protocol (UDP) [Pos80] as done, e.g., in Google's Quick UDP Internet Connections (QUIC) protocol [QUI], we deem these approaches orthogonal or unrelated to our work. In such cases, a reliable and ordered stream transmission can be implemented *non-cryptographically* either by TCP-like preprocessing of the UDP datagrams before handing them over to a stream-based channel according to our definition or by postprocessing UDP datagrams which are encrypted and authenticated individually (e.g., using an AEAD scheme).

flag.[48]

In our definition below for any message fragment m processed by Send we denote by c the (potentially empty) resulting ciphertext. We stress that c should not be interpreted as an encapsulation of m (i.e., we do not require that c decrypts to m, as we expand later) but just as a label for the output of Send on input m and the current state. In particular, letting Send output empty ciphertexts allows the algorithm to buffer message fragment for later sending. Similar considerations hold for Recv, which may buffer ciphertext fragments before returning a non empty message fragment. Figure 10.2 illustrates the behavior of the sending and receiving algorithms of a stream-based channel.

We proceed by defining syntax and correctness of stream-based channels.

Definition 10.1 (Syntax of stream-based channels)**.** *A* stream-based channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *with associated sending and receiving state space $\mathcal{S}_S$ resp. $\mathcal{S}_R$ and error space $\mathcal{E}$ with $\mathcal{E} \cap \{0,1\}^* = \emptyset$ consists of three efficient algorithms defined as follows.*

- $\mathsf{Init}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_{S,0}, \mathsf{st}_{R,0})$*. On input a security parameter 1^λ, this probabilistic algorithm outputs initial states $\mathsf{st}_{S,0} \in \mathcal{S}_S$, $\mathsf{st}_{R,0} \in \mathcal{S}_R$ for the sender and the receiver, respectively.*
- $\mathsf{Send}(\mathsf{st}_S, m, f) \xrightarrow{\$} (\mathsf{st}'_S, c)$*. On input a sending state $\mathsf{st}_S \in \mathcal{S}_S$, a message fragment $m \in \{0,1\}^*$, and a flush flag $f \in \{0,1\}$, this (possibly) probabilistic algorithm outputs an updated state $\mathsf{st}'_S \in \mathcal{S}_S$ and a ciphertext fragment $c \in \{0,1\}^*$.*
- $\mathsf{Recv}(\mathsf{st}_R, c) \to (\mathsf{st}'_R, m)$*. On input a receiving state $\mathsf{st}_R \in \mathcal{S}_R$ and a ciphertext fragment $c \in \{0,1\}^*$, this deterministic algorithm outputs an updated state $\mathsf{st}'_R \in \mathcal{S}_R$ and a message fragment (possibly containing error symbols) $m \in (\{0,1\} \cup \mathcal{E})^*$.*

Shorthand notation. Given a pair of states $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0})$, an integer $\ell \geq 0$, and tuples of message fragments $\mathbf{m} = (m_1, \ldots, m_\ell) \in (\{0,1\}^*)^\ell$ and of flush flags $\mathbf{f} = (f_1, \ldots, f_\ell) \in \{0,1\}^\ell$, let $(\mathsf{st}_S, \mathbf{c}) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}_{S,0}, \mathbf{m}, \mathbf{f})$ be shorthand for the sequential execution $(\mathsf{st}_{S,1}, c_1) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}_{S,0}, m_1, f_1), \ldots, (\mathsf{st}_{S,\ell}, c_\ell) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}_{S,\ell-1}, m_\ell, f_\ell)$ with $\mathbf{c} = (c_1, \ldots, c_\ell)$ and $\mathsf{st}_S = \mathsf{st}_{S,\ell}$. For $\ell = 0$ we define $\mathbf{c}$ to be the empty vector and $\mathsf{st}_{S,\ell} = \mathsf{st}_S$ to be the initial state. We use an analogous notation for the receiver's algorithm.

Correctness of stream-based channels should guarantee that if, after initialization, Send is fed with a message stream, and (a prefix of) the corresponding ciphertext stream is then processed by Recv, then no matter how the ciphertexts are fragmented at the sender's side and re-fragmented at the receiver's side (provided that the order of the bits is preserved), then the returned message stream is a prefix of the initial message stream. Moreover, when Recv consumes a full ciphertext fragment generated by a call to Send with the flush flag set to 1, the stream output by Recv should contain all the message fragments input to Send up to that call.

More precisely, if the receiver obtains (in an arbitrarily fragmented way) a prefix $\|\mathbf{c}'$ of the string of ciphertexts $\|\mathbf{c}$ created by the sender for an input message vector $\mathbf{m} \in (\{0,1\}^*)^\ell$ and flush flag vector $\mathbf{f} \in \{0,1\}^\ell$, and if string $\|\mathbf{c}'$ contains the concatenation $c_1\| \ldots \|c_i$ of the first i ciphertexts of $\mathbf{c}$, then we require that the message string $\|\mathbf{m}'$ returned on the receiver's side contains as a prefix the concatenation $m_1\| \ldots \|m_i$ of the first i messages of $\mathbf{m}$ for all indices $i \in \{0\} \cup \{j : f_j = 1\}$ for which the corresponding call to Send flushed its buffer (if all flush flags f_j are set to zero then the above concatenations of ciphertexts and messages are empty and the correctness condition is trivially fulfilled). In particular, this requires that the receiver must output the full and correct message stream if the last Send call has the flush flag f_ℓ set to 1.

[48] While any practical channel implementation will send buffered data out eventually (on explicit request or on timeouts), note that our model is general enough to also capture channels that do not offer any control over flushing: for this it suffices to consider a restricted channel interface where the flush flag is fixed to $f = 0$.

Definition 10.2 (Correctness of stream-based channels)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel. We say that* Ch *provides* correctness *if for all state pairs* $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0}) \xleftarrow{\$} \mathsf{Init}(1^\lambda)$*, all* $\ell, \ell' \geq 0$*, all choices of the randomness for algorithms* Init *and* Send*, all message-fragment vectors* $\mathbf{m} \in (\{0,1\}^*)^\ell$*, all flush-flag vectors* $\mathbf{f} \in \{0,1\}^\ell$*, all sending output sequences* $(\mathsf{st}_{S,\ell}, \mathbf{c}) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}_{S,0}, \mathbf{m}, \mathbf{f})$*, all ciphertext-fragment vectors* $\mathbf{c}' \in (\{0,1\}^*)^{\ell'}$*, and all receiving output sequences* $(\mathsf{st}'_{R,\ell'}, \mathbf{m}') \leftarrow \mathsf{Recv}(\mathsf{st}_{R,0}, \mathbf{c}')$*, we have for any* $i \in \{0\} \cup \{j : f_j = 1\}$ *that*

$$\|\mathbf{c}[1,\ldots,i] \preccurlyeq \|\mathbf{c}' \preccurlyeq \|\mathbf{c} \implies \|\mathbf{m}[1,\ldots,i] \preccurlyeq \|\mathbf{m}' \preccurlyeq \|\mathbf{m}.$$

Remark 10.3. Note that the receiver's output alphabet consists of bits and of distinct error symbols of the set $\mathcal{E}$; correctness therefore implies that the receiver does not output error symbols for genuine ciphertext streams.

Remark 10.4. It is instructive to compare our correctness definition with that of Boldyreva et al. [BDPS12]. There, correctness requires that if a sequence $\mathbf{m}$ of discrete messages is encrypted, and the resulting ciphertext stream $\|\mathbf{c}$ is then decrypted (possibly in a fragmented manner), then the obtained message sequence (when message separators ¶ are removed) is identical to the original sequence $\mathbf{m}$. In the special case of a single message, this implies that encryption 'always flushes' in the setting of [BDPS12], and is in turn the reason why encryption is necessarily an atomic operation. By contrast, in our setting the Send algorithm is equipped with a flush flag and, when the latter is set to zero, potentially the entire message fragment is buffered for delayed sending. This is, then, an essential difference between the setting of Boldyreva et al. [BDPS12] and the streaming one. An additional difference is that the correctness condition in [BDPS12] is stronger than ours as it incorporates a certain amount of robustness. More specifically, the sequence of ciphertext fragments $\mathbf{c}'$ submitted for decryption in the correctness definition of [BDPS12] may extend the sequence produced by encryption (in other words, $\|\mathbf{c}$ is only required to be a prefix of $\|\mathbf{c}'$ in order for decryption to still work correctly up to $\|\mathbf{c}$). Such cases will be dealt with by our integrity notions.

10.3 Security of Stream-Based Channels

In this section we introduce confidentiality and integrity notions for stream-based channels and study the relations among these notions. Our notions are game-based and extend security properties for stateful authenticated encryption [BKN04] (cf. Section 9.3) to the streaming setting.

10.3.1 Confidentiality

Following the approach of Bellare, Kohno, and Namprempre [BKN04] for stateful authenticated encryption we define confidentiality in terms of left-or-right indistinguishability of ciphertexts (cf. Section 9.3.1). We recall that chosen-plaintext attacks (CPA) for stateful encryption are modeled through a game in which $\mathcal{A}$ is given a left-or-right oracle $\mathcal{O}_{\mathsf{LoR}}$ that, upon being queried on pairs of messages (m_0, m_1), returns encryptions of either the 'left' messages m_0 or the 'right' messages m_1, depending on a secret bit $b \in \{0,1\}$. The game to model chosen-ciphertext attacks (CCA) is similar but additionally provides the adversary with a decryption oracle $\mathcal{O}_{\mathsf{Recv}}$ that $\mathcal{A}$ can query on ciphertexts of its choice, obtaining the corresponding decrypted messages except for *in-sync* ciphertexts (i.e., the sequence of ciphertexts output by the left-or-right oracle $\mathcal{O}_{\mathsf{LoR}}$ whose decryption would reveal the challenge sequence m_b by correctness). In both games $\mathcal{A}$'s goal is to determine the bit b. We now adapt these notions to the stream-based setting.

As for the case of symmetric encryption supporting ciphertext fragmentation—introduced by Boldyreva et al. in [BDPS12]—our security notions should reflect that the algorithms of a

stream-based channel support processing of arbitrary fragments of the message stream and of the ciphertext stream respectively. However, while Boldyreva et al. consider only fragmented decryption (i.e., the encryption process is atomic) and therefore focus their attention on the CCA setting, in our case the fragmentation at the sender also affects the adversarial capabilities in the CPA setting. We hence define two new indistinguishability notions, one for *chosen plaintext-fragment attacks* (IND-CPFA) and one for *chosen ciphertext-fragment attacks* (IND-CCFA). The corresponding experiments, that we denote by $\mathsf{Expt}^{\mathsf{IND\text{-}ATK},b}_{\mathsf{Ch},\mathcal{A}}$ for $\mathsf{ATK} \in \{\mathsf{CPFA}, \mathsf{CCFA}\}$, are depicted in Figure 10.3.

Before specifying in detail the logic of our experiments we introduce some useful notation. Within the experiments we denote by C_S the concatenation of the ciphertext fragments sent so far; similarly, we write C_R for the concatenation of the ciphertext fragments queried at the receiver. We abbreviate C_S and C_R as the sent (ciphertext) stream and the received (ciphertext) stream, respectively. We say that the sent and received streams are in-sync, or equivalently that they match, if the latter is a prefix of the former ($C_R \preccurlyeq C_S$). If $C_S \not\preccurlyeq C_R$ and $C_R \not\preccurlyeq C_S$ then we say that the two streams deviate, or equivalently that they are out-of-sync. In the experiment we keep a flag sync to indicate whether the above two streams are in-sync ($\mathsf{sync} = 1$) or not. As soon as the streams go out-of-sync we also say that also the oracle $\mathcal{O}_{\mathsf{Recv}}$ goes out-of-sync.

Adapting the CPA experiment to the stream-based settings is relatively easy. We do so by incorporating the flush flag f into the oracle $\mathcal{O}_{\mathsf{LoR}}$ and letting the adversary also specify the value of f; this provides $\mathcal{A}$ with the same interface for Send as that of an application. For each query (m_0, m_1, f) the oracle $\mathcal{O}_{\mathsf{LoR}}$ operates as follows: after checking that the message fragments m_0 and m_1 have the same bit length, it invokes Send on input the current state st_S, message m_b, and flush flag f, it records the resulting ciphertext fragment c into the ciphertext stream C_S, and finally gives c to the adversary.

Formalizing a sound security notion for the CCA setting, where the adversary can also obtain the output of Recv for chosen ciphertext-fragments, turns out to be more delicate. Ideally, we envision a receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ that lets $\mathcal{A}$ see as much decrypted plaintext as possible without enabling trivial attacks. Following this principle we mimic the strategy of Bellare et al. [BKN04] to model stateful encryption security by declaring $\mathcal{O}_{\mathsf{Recv}}$ to be in-sync—thus instructing it to artificially suppress the output of Recv—as long as the adversary supplies a prefix of the original ciphertext stream output by the left-or-right oracle.

There is, however, a definitional challenge to surmount. In contrast to stateful encryption where messages input to the encryption algorithm are considered as atomic units and correspond in a one-to-one manner to the output ciphertexts, in the streaming scenario messages and ciphertexts can be arbitrarily fragmented. Therefore, it is not clear a priori how to translate deviations of the ciphertext sequences into deviation of the message sequences. To adapt the suppression mechanism of Bellare et al. to the streaming setting we need to determine at which point exactly the ciphertext stream at the receiver should be considered out-of-sync.

Suppression mechanism for symmetric encryption supporting fragmentation. The experiment for indistinguishability against chosen-fragment attacks (IND-sfCFA) proposed by Boldyreva et al. [BDPS12, Definition 4] in the context of symmetric encryption supporting ciphertext fragmentation stays close to the original definitions of [BKN04] and conservatively defines synchronization to be lost *at the ciphertext boundaries*. That is, $\mathcal{O}_{\mathsf{Recv}}$ suppresses the output of Recv up to the longest sequence of genuine ciphertexts (recall that in [BDPS12] the encryption algorithm is atomic, and hence it outputs entire ciphertexts rather than fragments, making it possible to identity ciphertext boundaries in a security experiment). In a run of the IND-sfCFA experiment let $m_1, \ldots, m_i$ be the messages processed by Send and let $c_1, \ldots, c_i$ be the resulting ciphertexts, let $j \leq i$ be maximal such that the receiving algorithm processes entirely the

sequence of the first j sent ciphertexts, and suppose that the stream C_R of ciphertext fragments processed by Recv deviates from the genuine sequence $C_S = c_1 \| \cdots \| c_i$. Then synchronization is considered to be lost from the first bit following ciphertext c_j, and hence $\mathcal{O}_{\mathsf{Recv}}$ suppresses exactly the first j sent messages $m_1, \ldots, m_j$ from the output of Recv.

As we show in the following examples, this option is inappropriate in a stream-based setting where the ciphertext fragments output by Send do not necessarily correspond one-to-one to the input message-fragments.

Consider the case of TLS and the Send algorithm being called on a $(2^{14} + 1)$-byte input message with the flush flag set to 1—mimicking the behavior of many TLS implementations that keep no send buffer. Obeying the limit of at most 2^{14} bytes of payload in a single TLS record, Send is forced to output a ciphertext fragment containing (at least) two TLS records. According to the IND-sfCFA experiment in [BDPS12], an adversary could forward this fragment to the decryption oracle with the second record modified but the first record untouched. The adversary would then obtain the decryption of *both* records as the IND-sfCFA experiment considers them as jointly forming a *single* ciphertext, hence revealing parts of the challenge message string. Thus, the IND-sfCFA notion would be unachievable for TLS.

As another example assume that a stream-based channel aiming only at confidentiality generates the ciphertext stream C_S as the bitwise XOR of the message stream M_S and the output of a stream cipher. In this setting, the IND-sfCFA notion would consider the ciphertext fragments output by Send on the input message fragments to be units that should be either completely kept confidential or, if modified, can be fully leaked to the adversary. However, with the channel exhibiting no further structure, enforcing any block boundaries clearly becomes artificial.

Suppression mechanism for stream-based channels. Taking into account that in the streaming scenario the output of Send is a bit stream without any further structure in general, we declare synchronization to be lost starting from the first bit of the receiving ciphertext stream C_R deviating from the genuine stream C_S.

Concretely, suppose that during an execution of the IND-CCFA experiment the adversary causes C_R to deviate from C_S by submitting for decryption a *strict* prefix of the genuine stream followed by additional bits that deviate from C_S (using a more compact notation: $[C_S, C_R] \prec C_S$ and $[C_S, C_R] \prec C_R$). In this case the streams C_S and C_R are considered to be in-sync up to their longest common prefix $[C_S, C_R]$ while the deviating portion $C_R \,\%\, [C_S, C_R]$ is out-of-sync. Given this, we let $\mathcal{O}_{\mathsf{Recv}}$ process ciphertext fragments submitted for decryption by updating C_R with the newly queried fragment and, as long as C_R is a prefix of C_S, invoking Recv (thereby updating the state st_R) on this fragment and suppressing the corresponding output. If the adversary instead submits a fragment that causes C_R to deviate from C_S the oracle stops suppressing and, from then on, the output of Recv is given to the adversary.

Processing the first ciphertext fragment that causes a deviation requires some care, though. The challenging situation is as follows: all fragments submitted for decryption so far are in-sync ($C_R \preccurlyeq C_S$) and the 'next' ciphertext fragment c induces a deviation ($C_R \| c \not\preccurlyeq C_S$). Now, if c contains a non-empty in-sync prefix its decryption may trivially reveal challenge bits that should instead be suppressed (as in the above example concerning TLS). To resolve this issue we let the receiving oracle invoke Recv on both the entire fragment c and its longest in-sync prefix $\widetilde{c}$ (the latter is the longest prefix of c that matches C_S), making sure that Recv takes as input the same state st_R for both operations. Let m and $\widetilde{m}$ be the resulting message fragments respectively, and note that $\widetilde{m}$ matches the genuine message stream by correctness. Then $\mathcal{O}_{\mathsf{Recv}}$ suppresses from m its longest prefix that matches the genuine stream (i.e., it suppresses the potentially empty fragment $[m, \widetilde{m}]$) and gives the resulting string $m' = m \,\%\, [m, \widetilde{m}]$ to the adversary. The

$\underline{\mathsf{Expt}^{\mathsf{IND\text{-}ATK},b}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)}$:
1 $(\mathsf{st}_S, \mathsf{st}_R) \xleftarrow{\$} \mathsf{Init}(1^\lambda)$
2 $\mathsf{sync} \leftarrow 1$
3 $C_S \leftarrow \varepsilon$, $C_R \leftarrow \varepsilon$
4 $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}},[\mathcal{O}_{\mathsf{Recv}}]_{\mathsf{ATK=CCFA}}}(1^\lambda)$
5 return b'

$\underline{\mathcal{O}_{\mathsf{LoR}}(m_0, m_1, f)}$:
6 if $|m_0| \neq |m_1|$ then
7 return $\lightning$
8 $(\mathsf{st}_S, c) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}_S, m_b, f)$
9 $C_S \leftarrow C_S \| c$
10 return c

$\underline{\mathcal{O}_{\mathsf{Recv}}(c)}$:
11 if $\mathsf{sync} = 0$ then // already out-of-sync
12 $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
13 return m
14 else if $C_R \| c \preccurlyeq C_S$ then // still in-sync
15 $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
16 $C_R \leftarrow C_R \| c$
17 return ε
18 else
19 if $C_R \prec [C_R \| c, C_S]$ then
// c deviates or exceeds, contains genuine part
20 $\widetilde{c} \leftarrow [C_R \| c, C_S] \,\%\, C_R$
21 $\widetilde{\mathsf{st}_R} \leftarrow \mathsf{st}_R$
22 $(\widetilde{\mathsf{st}_R}, \widetilde{m}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}_R}, \widetilde{c})$
23 $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
24 $m' \leftarrow m \,\%\, [m, \widetilde{m}]$
25 else // c deviates or exceeds, contains no genuine part
26 $(\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
27 if $C_S \not\preccurlyeq C_R \| c$ or $m' \neq \varepsilon$ then
// deviation, or exceeding portion produces output
28 $\mathsf{sync} \leftarrow 0$
29 $C_R \leftarrow C_R \| c$
30 return m'

Figure 10.3: Security experiment for *confidentiality* ($\mathsf{IND\text{-}ATK}$ with $\mathsf{ATK} \in \{\mathsf{CPFA}, \mathsf{CCFA}\}$) of stream-based channels. The brackets $[\mathcal{O}_{\mathsf{Recv}}]_{\mathsf{ATK=CCFA}}$ indicate that only the $\mathsf{IND\text{-}CCFA}$ adversary has access to the $\mathcal{O}_{\mathsf{Recv}}$ oracle.

idea behind this strategy is that any potential challenge bit originating from the in-sync part of c remains hidden from the adversary.

Another subtlety arising from the ciphertext fragmentation concerns the possibility that C_R and C_S are neither in-sync nor out-of-sync. This may happen if the adversary submits to $\mathcal{O}_{\mathsf{Recv}}$ the *entire* (potentially empty) genuine stream of ciphertexts followed by additional bits, making C_R exceed C_S. In such a case the stream C_R does not explicitly deviate from, but only extends, the genuine stream C_S, that is, $C_S \prec C_R$.

One could argue that submitting for decryption any bit that has not been honestly produced at the sender should be considered an active attack and, thus, any part of C_R exceeding C_S should be declared out-of-sync.[49] This intuition turns out to be wrong, though. In fact, the ability to guess a small prefix of the 'next' ciphertext fragment output by Send should not be considered as giving a significant advantage to the adversary. Then, declaring synchronization to be lost with the first bit of C_R exceeding C_S would allow for trivial attacks like the following. The adversary starts by making a guess on the first bit $d' \in \{0, 1\}$ output by Send and querying $\mathcal{O}_{\mathsf{Recv}}$ on input fragment d'. As $C_R = d'$ deviates from $C_S = \varepsilon$ this first query desynchronizes the receiving oracle. The adversary proceeds by posing a left-or-right query $(m_0, m_1, 1)$ with $m_0 \neq m_1$, obtains a challenge ciphertext-fragment c', and if its guess was correct (i.e., $d' \preccurlyeq c'$) it submits for decryption the ciphertext fragment $c' \,\%\, d'$, otherwise it terminates. If adversary correctly guessed the first bit of c' then by correctness it gets from $\mathcal{O}_{\mathsf{Recv}}$ the message $m' = m_b$. This strategy succeeds with probability $\frac{1}{2}$. Merely guessing a small prefix of the next genuine ciphertext fragment which does not produce any output is possible for any stream-based channel, but should not be considered as a success of the adversary.

[49] This was indeed the argument adopted in [BDPS12] as well as in the proceedings version of this work [FGMP15], and later identified as flawed by Degabriele [Deg16] in May 2016.

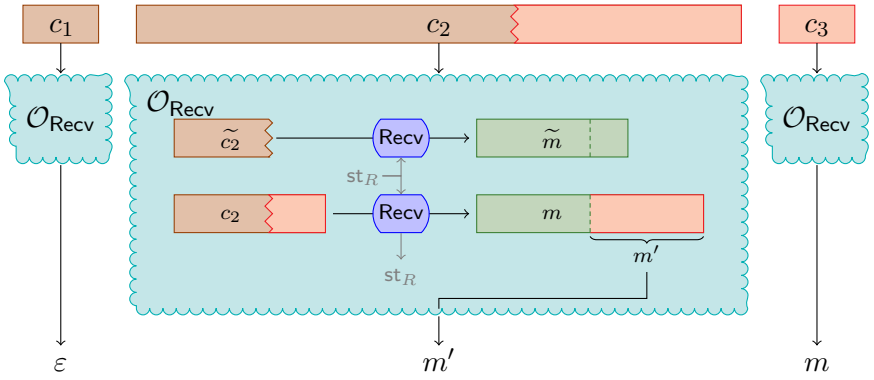


Figure 10.4: Illustration of the $\mathcal{O}_{\mathsf{Recv}}$ oracle behavior in the IND-CCFA experiment from Figure 10.3 for ciphertext fragment inputs c_1, c_2, c_3 specified by the adversary $\mathcal{A}$ where c_2 deviates from or exceeds the genuine ciphertext stream after the zigzag line. The oracle $\mathcal{O}_{\mathsf{Recv}}$ goes out-of-sync on deviation or if exceeding ciphertext input produces non-empty output.

To exclude the class of attacks like the one just described we adopt the following strategy: we only declare synchronization to be lost when the exceeding portion $C_R \,\%\, C_S$ produces a *non-empty* output—in which case we also conservatively provide the adversary with this output. In other words, we only give credit to the adversary if it is able to predict a non-trivial ciphertext fragment that actually leads to a non-empty plaintext. This allows $\mathcal{O}_{\mathsf{Recv}}$ to later suppress further message bits in case, as a consequence of the adversary posing more left-or-right queries, C_S and C_R end up being matching again (in the sense that $C_R \preccurlyeq C_S$).

To facilitate understanding we illustrate in Figure 10.4 how $\mathcal{O}_{\mathsf{Recv}}$ processes the first deviating ciphertext fragment ($C_R\|c \not\preccurlyeq C_S$) or an exceeding ciphertext fragment ($C_S \prec C_R\|c$) by performing two calls to Recv, one on input c and the other on input $\widetilde{c}$.

Putting everything together, we can now unpack from Figure 10.3 the instructions executed by the $\mathcal{O}_{\mathsf{Recv}}$ oracle upon being queried on a ciphertext fragment c. Its logic treats the two simplest cases first. In case synchronization has been already lost (indicated by the flag sync being 0, line 11) the oracle responds with the entire output of Recv on input the fragment c. If c is in-sync ($C_R\|c \preccurlyeq C_S$, line 14) the oracle invokes Recv on input c but fully suppresses its output. Next consider the case in which c makes C_R extend C_S ($C_R\|c \not\preccurlyeq C_S$ but $C_S \preccurlyeq C_R\|c$) or causes a deviation ($C_R\|c \not\preccurlyeq C_S$ and $C_S \not\preccurlyeq C_R\|c$). Here the oracle $\mathcal{O}_{\mathsf{Recv}}$ first checks whether c contains a non-empty, in-sync prefix $\widetilde{c}$ matching C_S which may contain challenge-message bits that should be suppressed. Note that the corresponding check in line 19 evaluates to true if either $C_R\|c$ deviates from C_S but c contains a non-empty in-sync prefix $\widetilde{c}$ (thus $[C_R\|c, C_S] = C_R\|\widetilde{c} \prec C_S$ with $\widetilde{c} \neq \varepsilon$), or $C_R\|c$ extends C_S but c contains a non-empty prefix matching C_S (i.e., $[C_R\|c, C_S] = C_R\|\widetilde{c} = C_S$ also with $\widetilde{c} \neq \varepsilon$). If from the above check it turns out that c contains a non-empty in-sync prefix, the receiving operation is handled by performing a double invocation of Recv on input identical states $\mathsf{st}_R = \widetilde{\mathsf{st}_R}$ and ciphertext fragments c and $\widetilde{c}$ respectively, which yields message fragments m and $\widetilde{m}$, as described earlier (and illustrated in Figure 10.4). We stress that the second invocation should be considered as an auxiliary step performed by the oracle to establish which portion of m, if at all, shall be suppressed (line 24). Indeed, the latter step is skipped in case c is fully deviating from or fully exceeding C_S (line 25). In either case, the oracle proceeds with determining whether synchronization is lost upon the receipt of c because of a deviation ($C_S \not\preccurlyeq C_R\|c$) or because the adversary managed to guess a non-trivial ciphertext fragment leading to a non-emtpy output ($C_S \preccurlyeq C_R\|c$ but $m' \neq \varepsilon$). The message fragment m' here denotes the output of Recv corresponding to the deviating or exceeding part of c; this is the actual message fragment that $\mathcal{O}_{\mathsf{Recv}}$ returns to the adversary.

We are now ready to give the formalism of our confidentiality experiments.

Definition 10.5 (IND-CPFA and IND-CCFA security)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and experiment* $\mathsf{Expt}^{\mathsf{IND\text{-}ATK},b}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *and a bit* b *be defined as in Figure 10.3, where* ATK *is a placeholder for either* CPFA *or* CCFA*.*

We say that Ch *provides* indistinguishability under chosen plaintext-fragment attacks*, respectively* chosen ciphertext-fragment attacks *(*IND-CPFA*, resp.* IND-CCFA*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}ATK}}_{\mathsf{Ch},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}ATK},1}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}ATK},0}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right]\right|.$$

Length hiding. Note that we do not consider the extended length-hiding setting that was introduced in [PRS11] to model TLS's variable length padding capability and subsequently incorporated into the ACCE security definition for secure channels in [JKSS12, KPW13]. While our work could conceivably be extended to incorporate length hiding, it remains unclear to us what its value would be in the setting of *streaming* channels, since length hiding is a notion intrinsically connected to atomic messages.

On the scope of our confidentiality definition. When formalizing a security goal it is common to develop a notion that is as strong as possible, yet achievable. The inability to foresee new attacks is the main reason for aiming at the strongest notion. However, sometimes this conservative approach leads to security notions that are 'too strong' for some applications, meaning that some schemes which have no actual vulnerability are declared insecure within the model. An example of a security notion that—one might argue—is too strong is IND-CCA security for symmetric encryption. For instance, if one starts with an IND-CCA-secure encryption scheme and modifies it by letting the encryption routine append a redundant bit to each ciphertext and letting the decryption routine ignore that last bit of each ciphertext, the resulting scheme is no longer IND-CCA-secure. However, adding a redundant bit that is then ignored for decryption should not harm the scheme's confidentiality.[50]

Our IND-CCFA notion for stream-based channel may likewise be too strong for some applications. In the extended full version of our work [FGMP17] we describe a scheme due to Poettering [Poe16] that, despite being intuitively confidential, is vulnerable to an IND-CCFA attack. Briefly, this stream-based channel processes message fragments to be sent by AEAD-encrypting fixed-length chunks of each input fragment, and similarly processes ciphertext fragments to be received by AEAD-decrypting corresponding ciphertext blocks in such a way that, if any of the AEAD decryptions fails, then a distinguished constant string is returned rather than an error. Clearly, the scheme does not provide integrity protection, because the receiver would not be able to detect that the message output stems from an error. However, because of the AEAD security, the message fragment output by Recv on input a deviating ciphertext fragment should only reveal the distinguished string independently of the challenge-message fragment and, thus, confidentiality should not be compromised. The way message output is suppressed in our IND-CCFA experiment (cf. Figure 10.3) however enables an IND-CCFA attack on this scheme which is always successful by checking for common prefixes between the challenge messages and the distinguished constant error string (see [FGMP17] for the details). Exploring possible relaxations of the stream-based IND-CCFA confidentiality notion which still uphold an intuitive, strong confidentiality guarantee is an interesting direction for future work.

[50] An alternative notion of confidentiality that precisely aims at resolving this issue was proposed by Canetti et al. [CKN03] as RCCA security (for 'replayable' CCA) in the public-key setting (and can be easily extended to the secret-key setting).

10.3.2 Integrity

Next, we formalize integrity notions for stream-based channels. We highlight that, while integrity properties for atomic messages (and atomic ciphertexts) are well-understood, no previous work considered integrity in the non-atomic setting. In particular Boldyreva et al. [BDPS12] only addressed confidentiality in the presence of ciphertext fragmentation; their notions were later and concurrently to our work augmented with integrity notions by Albrecht et al. [ADHP16]. We define integrity notions for stream-based channels as refinements of standard (stateful) properties of plaintext integrity (INT-sfPTXT) resp. ciphertext integrity (INT-sfCTXT) from [BKN04] (cf. Section 9.3.1) and refer to the new properties as *plaintext-stream integrity* resp. *ciphertext-stream integrity* (INT-PST resp. INT-CST).

Similarly to the setting with atomic messages, INT-PST ensures that no adversarial query to the receiving oracle causes the message stream output by Recv to deviate from the message stream input to Send. This notion is quite simple to formulate. Formalizing the stronger INT-CST property demands more care. Intuitively, from ciphertext integrity we expect that when processing any 'out-of-sync' ciphertext, the algorithm Recv should return an error message. However, when considering a stream-based interface it may happen that Recv processes an out-of-sync ciphertext which does not yet contain 'enough information' to be recognized as being invalid; in this case the receiving algorithm would buffer (part of) the ciphertext and wait for further fragments until a sufficiently long ciphertext string is available to be processed and deemed as valid or invalid. In such a scenario, a naive adaptation of the INT-sfCTXT definition of [BKN04] would allow trivial attacks by declaring successful any adversary that makes the Recv buffer (part of) an out-of-sync ciphertext, without producing non-trivial output. Our notion of ciphertext-stream integrity carefully identifies the case just described and, by letting the receiving oracle wait for further ciphertext fragments, declares the adversary successful only if Recv outputs a non-empty message fragment resulting from an out-of-sync or exceeding portion of the ciphertext stream.

We formalize integrity of plaintext and ciphertext streams through the security experiment $\mathsf{Expt}^{\mathsf{INT\text{-}ATK}}_{\mathsf{Ch},\mathcal{A}}$ depicted in Figure 10.5. The experiment provides the adversary with oracles $\mathcal{O}_{\mathsf{Send}}$ and $\mathcal{O}_{\mathsf{Recv}}$, where the former grants $\mathcal{A}$ access to algorithm Send under arbitrarily chosen message fragments and the latter gives $\mathcal{A}$ an interface with algorithm Recv. We highlight that, while the sending oracle $\mathcal{O}_{\mathsf{Send}}$ is common for both experiments INT-PST and INT-CST, the receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ follows different procedures in the two cases, as we further explain below.

In the execution of the INT-PST experiment, $\mathcal{O}_{\mathsf{Send}}$ maintains in string M_S the stream of all sent message fragments and, analogously, $\mathcal{O}_{\mathsf{Recv}}$ maintains in M_R the stream of all received message fragments (and/or error symbols). The adversary wins the game if it causes M_S and M_R to deviate in such a way that their difference contains more than error symbols. Formally, we demand that the string M_R output by the receiver is not a prefix of the sender's string M_S, but such that this prefix-freeness is not only due to error symbols from $\mathcal{E}$.

In the INT-CST experiment oracles $\mathcal{O}_{\mathsf{Send}}$ and $\mathcal{O}_{\mathsf{Recv}}$ maintain strings C_S and C_R to record the streams of sent ciphertexts, resp. received ciphertext fragments. Furthermore, $\mathcal{O}_{\mathsf{Recv}}$ decides when the adversary wins by inspecting sent and received *ciphertext* streams, an inherently more complex task than looking for deviations in the underlying sequences of sent/received message fragments. Indeed, in a stream-based channel the algorithm Recv may need to buffer several ciphertexts before being able to recover the underlying message stream or detecting that an error occurred; such a behavior is reflected in our experiment. When processing in-sync ciphertexts $\mathcal{O}_{\mathsf{Recv}}$ simply appends each new fragment to C_R. At the moment when an out-of-sync ciphertext fragment or one that exceeds the sent ciphertext stream arrives, the oracle compares the outputs of algorithm Recv when processing (i) the current input ciphertext c and (ii) its longest in-sync

$\underline{\mathsf{Expt}^{\mathsf{INT\text{-}ATK}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)}$:
1 $(\mathsf{st}_S, \mathsf{st}_R) \xleftarrow{\$} \mathsf{Init}(1^\lambda)$
2 $\mathsf{sync} \leftarrow 1$
3 $\mathsf{win} \leftarrow 0$
4 $M_S, C_S \leftarrow \varepsilon$, $M_R, C_R \leftarrow \varepsilon$
5 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}}, \mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$
6 return win

$\underline{\mathcal{O}_{\mathsf{Send}}(m, f)}$:
7 $(\mathsf{st}_S, c) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}_S, m, f)$
8 $M_S \leftarrow M_S \| m$
9 $C_S \leftarrow C_S \| c$
10 return c

$\underline{\mathsf{INT\text{-}PST}\ \mathcal{O}_{\mathsf{Recv}}(c)}$:
11 $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
12 $M_R \leftarrow M_R \| m$
13 if $M_R \,\%\, [M_R, M_S] \notin \mathcal{E}^*$ then
14 $\quad \mathsf{win} \leftarrow 1$
15 return m

$\underline{\mathsf{INT\text{-}CST}\ \mathcal{O}_{\mathsf{Recv}}(c)}$:
16 if $\mathsf{sync} = 0$ then // already out-of-sync
17 $\quad (\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
18 $\quad$ if $m \notin \mathcal{E}^*$ then $\mathsf{win} \leftarrow 1$
19 else if $C_R \| c \preccurlyeq C_S$ then // still in-sync
20 $\quad (\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
21 $\quad C_R \leftarrow C_R \| c$
22 else
23 $\quad$ if $C_R \prec [C_R \| c, C_S]$ then
$\quad\quad$ // c deviates or exceeds, contains genuine part
24 $\quad\quad \widetilde{c} \leftarrow [C_R \| c, C_S] \,\%\, C_R$
25 $\quad\quad \widetilde{\mathsf{st}_R} \leftarrow \mathsf{st}_R$
26 $\quad\quad (\widetilde{\mathsf{st}_R}, \widetilde{m}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}_R}, \widetilde{c})$
27 $\quad\quad (\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
28 $\quad\quad m' \leftarrow m \,\%\, [m, \widetilde{m}]$
29 $\quad$ else // c deviates or exceeds, contains no genuine part
30 $\quad\quad (\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
31 $\quad\quad m \leftarrow m'$
32 $\quad$ if $C_S \not\preccurlyeq C_R \| c$ or $m' \neq \varepsilon$ then
$\quad\quad$ // deviation, or exceeding portion produces output
33 $\quad\quad \mathsf{sync} \leftarrow 0$
34 $\quad C_R \leftarrow C_R \| c$
35 $\quad$ if $m' \notin \mathcal{E}^*$ then $\mathsf{win} \leftarrow 1$
36 return m

Figure 10.5: Security experiment for *integrity* ($\mathsf{INT\text{-}ATK}$ with $\mathsf{ATK} \in \{\mathsf{PST}, \mathsf{CST}\}$) of stream-based channels. A PST-attacker is provided with access to the left $\mathcal{O}_{\mathsf{Recv}}$ oracle (INT-PST), whereas a CST-attacker is instead granted access to the oracle on the right-hand side (INT-CST).

prefix $\widetilde{c}$. The adversary wins if $\mathcal{O}_{\mathsf{Recv}}$ outputs more in case (i) than it would in case (ii) and if the difference between the two outputs is a non-empty, valid message. It also wins if it is able to make Recv output a non-empty, valid message with a subsequent out-of-sync ciphertext. As for confidentiality (see the discussion in Section 10.3.1) we consider the $\mathcal{O}_{\mathsf{Recv}}$ oracle to go out of sync (and set $\mathsf{sync} \leftarrow 0$) if the ciphertext fragment deviates from the corresponding bits in the sent ciphertext stream or, when it merely exceeds the sent stream, if the output of Recv for the exceeding part is non-empty.[51]

Definition 10.6 ($\mathsf{INT\text{-}PST}$ and $\mathsf{INT\text{-}CST}$ security)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and experiment* $\mathsf{Expt}^{\mathsf{INT\text{-}ATK}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 10.3, where* ATK *is a placeholder for either* PST *or* CST*.*

We say that Ch *provides* integrity of plaintext streams*, respectively* ciphertext streams *(*$\mathsf{INT\text{-}PST}$*, resp.* $\mathsf{INT\text{-}CST}$*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{INT\text{-}ATK}}_{\mathsf{Ch},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{INT\text{-}ATK}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right].$$

Remark 10.7. Our definitions of integrity do not preclude from deeming those channels secure in which message bits can be output as a result of the adversary delivering *partial* ciphertexts to

[51] As already discussed in Section 10.3.1, withholding synchronization loss for exceeding fragments that produce no output eliminates the trivial attack which was present in the proceedings version of this work [FGMP15]. There, the adversary could guess (and input to $\mathcal{O}_{\mathsf{Recv}}$) the first bit(s) of the next $\mathcal{O}_{\mathsf{Send}}$ output and, if successful, feed in the remaining ciphertext. With $\mathcal{O}_{\mathsf{Recv}}$ considered out-of-sync on these first bit(s), although generating no output, this attack illegitimately was considered a successful break of ciphertext-stream integrity.

the Recv oracle. This is because in the streaming setting we care about the adversary's ability to force the receiver to accept message fragments corresponding to a part of the ciphertext stream that has gone out-of-sync, without attaching importance to ciphertext boundaries. Hence, this is quite distinct from the usual atomic setting. Of course, applications making use of a streaming channel may wish to recover a secure channel for atomic messages, in a more traditional sense. For example, this is the case for HTTP running over TLS and, as noted in the introduction, has been a source of confusion for developers and led to concrete attacks on protocols such as TLS [SP13, BDF+14]. We will examine this situation in greater detail in Chapter 11.

We further note that stream-based integrity providing intuitively weaker guarantees than atomic-message integrity seems to be an intrinsic consequence of the nature of stream-based channels. In particular, apparent avenues for strengthening the given integrity definition lead to notions which are clearly inappropriate in the streaming setting. On the one hand, requiring a channel to output an error immediately after processing the first bit deviating from the sent ciphertext stream is, for most constructions, an unattainable goal as it is in general impossible to decide if an initial bit received is genuine or not.[52] On the other hand, requiring that a channel does not output any message bit until a full ciphertext output by Send is received inappropriately enforces an atomic structure on the channel, i.e., basically the one of [BDPS12]. The latter notion, as already discussed, is too strong for channels that, like TLS, might output ciphertexts which contain multiple, independent parts.

10.3.3 Relations Amongst Notions and Generic Composition Theorem

We now explore relations between confidentiality and integrity—well-established for atomic messages by [BN00, BKN04] and follow-up work, culminating in [NRS14]—and investigate whether these relations can be lifted to our streaming setting. We highlight that, since integrity for encryption schemes supporting ciphertext fragmentation was not addressed in [BDPS12], we are the first to consider such relations in the presence of fragmentation.

Ideally we would like to translate the classic implications for authenticated encryption $\mathsf{IND\text{-}CCA} \implies \mathsf{IND\text{-}CPA}$, $\mathsf{INT\text{-}CTXT} \implies \mathsf{INT\text{-}PTXT}$, and the powerful composition result $\mathsf{IND\text{-}CPA} \wedge \mathsf{INT\text{-}CTXT} \implies \mathsf{IND\text{-}CCA}$, all from [BN00], to the realm of stream-based channels. It is immediate to see that, as in the setting where messages are atomic, the stronger notions implies the weaker ones for both confidentiality and integrity individually. Unfortunately, when integrity and confidentiality are targeted simultaneously, the situation for streams is fundamentally more challenging.

Recall that, in the atomic-message setting, the proof of the composition theorem in [BN00] proceeds in two steps: starting from the IND-CCA game, one first bounds the probability that the adversary submits a valid decryption query distinct from an output of the encryption oracle by using the INT-CTXT advantage. This then allows a reduction to the IND-CPA experiment (now assuming integrity of ciphertexts), simply by answering all decryption queries with the distinguished error symbol $\perp$. As already noted by Boldyreva et al. [BDPS14], the same proof strategy does not work for schemes that have multiple decryption error symbols (which models common real-world behavior of encryption schemes). This is because the reduction can no longer (in general) know which one of the several possible error symbols should be output when simulating decryption.

Thus the classic result $\mathsf{IND\text{-}CPA} \wedge \mathsf{INT\text{-}CTXT} \implies \mathsf{IND\text{-}CCA}$ already does not follow in the situation where multiple error messages are possible, not even considering streaming. Worse, [BDPS14] shows that, in the multiple decryption error setting, there exist schemes that are secure in both IND-CPA and INT-CTXT senses, yet are not IND-CCA secure. We show later

[52] Indeed, stream-based integrity does not enforce that Recv outputs an error on deviating ciphertext fragments at all, but is also satisfied by a channel providing no output (i.e., the empty string) in such cases.

in this section that similar issues arise for stream-based channels, even when restricting to the case of single error messages. Specifically, fragmentation at the receiver's side makes it harder to emulate a receiving oracle for the IND-CCFA experiment given a receiving oracle for the INT-CST game.

As a remedy we propose an adapted version of the composition theorem, resurrecting the result both in our streaming setting and in the case of multiple errors that was considered in [BDPS14]. However this result can be proven only at the cost of introducing further assumptions on the output behavior of the receiving algorithm. The conditions for the composition theorem may initially look quite demanding but, as we confirm in Section 10.4, there exist natural schemes that satisfy the required conditions. Moreover, the use of the composition theorem is not the only route to achieving IND-CCFA security: for specific schemes it may be possible to prove IND-CCFA security directly.

Confidentiality. A study of the experiments in Figure 10.3 immediately shows that IND-CCFA security implies IND-CPFA security, since an attacker in the IND-CPFA game only needs to emulate the left-or-right oracle to provide a faithful simulation of the IND-CCFA game, and can trivially do so by relaying all encryption queries to its own left-or-right oracle.

Integrity. Assume that ciphertext-stream integrity (INT-CST) from Definition 10.6 holds for a stream-based channel. Then the channel also provides integrity of plaintext streams (INT-PST) and the security reduction is tight. To see why consider the integrity experiment depicted in Figure 10.5: given the INT-CST property, every efficient adversary either never produces a ciphertext stream C_R that deviates from the ciphertext stream C_S (hence, by correctness, no deviation will occur in the underlying message streams) or, if it generates a stream C_R that does deviate from C_S, by INT-CST the underlying message streams will only differ by an error string. We formalize this result in the following proposition.

Proposition 10.8. *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a correct stream-based channel which is* INT-CST *secure. Then the channel is also* INT-PST *secure. Furthermore,* $\mathbf{Adv}^{\mathsf{INT\text{-}PST}}_{\mathsf{Ch},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{A}}(\lambda)$ *for any algorithm* $\mathcal{A}$.

Proof. Assume that $\mathcal{A}$ attacks the INT-PST property of the channel. First note that $\mathcal{A}$ has the same interfaces as if attacking INT-CST such that we can think of running both experiments simultaneously. It then suffices to show that, if we set $\mathsf{win} \leftarrow 1$ in line 14 of the INT-PST experiment, then we would also set $\mathsf{win} \leftarrow 1$ (in line 18 or 35) in the simultaneous execution of the INT-CST experiment (both in Figure 10.5). Note that as long as $\mathsf{sync} = 1$ and the ciphertext stream submitted to $\mathcal{O}_{\mathsf{Recv}}$ is a prefix of the one created by $\mathcal{O}_{\mathsf{Send}}$, then the receiver's oracle in experiment INT-CST would indeed return the recovered message fragment, as in the INT-PST experiment.

Suppose that $\mathcal{A}$ triggers $\mathsf{win} \leftarrow 1$ in the INT-PST experiment. If at this point $C_R \preccurlyeq C_S$ then, because of correctness of the channel, we must also have $M_R \preccurlyeq M_S$, implying that win would not have been set. It follows that there must exist some $C_R \| c \not\preccurlyeq C_S$ where c is the first call to $\mathcal{O}_{\mathsf{Recv}}$ where the concatenation of the receiver strings deviate from or exceed the ciphertext stream of the sender. At this point we must also have $\mathsf{sync} = 1$ and $\mathsf{win} = 0$ in the INT-CST experiment and we enter the third case in line 22.

If in this case c contains some non-deviating prefix $\widetilde{c} \preccurlyeq c$, we compute $\widetilde{c}$ such that $C_R \| \widetilde{c} \preccurlyeq C_S$ is maximal. It again follows from correctness that for the processed $\widetilde{c}$ we get a message fragment $\widetilde{m}$ such that $M_R \| \widetilde{m} \preccurlyeq M_S$. But in order to set $\mathsf{win} \leftarrow 1$ in the INT-PST experiment, we must have that the full fragment c makes Recv output a message fragment m such that m contains message bits beyond the common prefix with M_S. It follows that $m' \leftarrow m \,\%\, [m, \widetilde{m}]$, the fragment

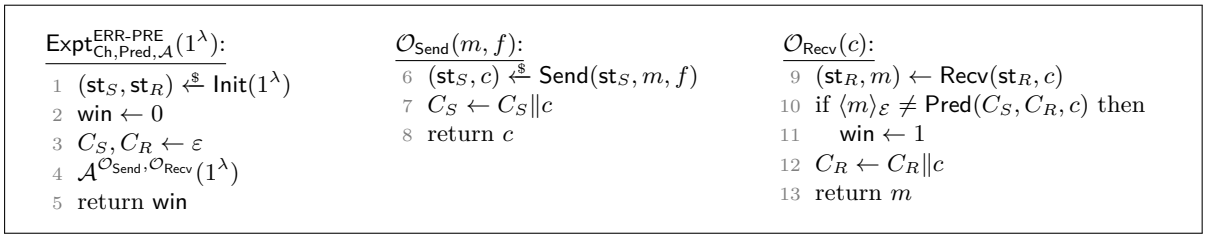


Figure 10.6: Security experiment for *error predictability* (ERR-PRE) of stream-based channels. We denote by $\langle \cdot \rangle_{\mathcal{E}} \colon (\{0,1\} \cup \mathcal{E})^* \to \mathcal{E}^*$ the 'projection onto the error space', i.e., the mapping that removes from a string all occurrences that do not belong to the error space $\mathcal{E}$. For instance, let $\mathcal{E} = \{\bot_1, \bot_2\}$ and $m = 01\bot_1 100\bot_2$; then $\langle m \rangle_{\mathcal{E}} = \bot_1\bot_2$.

of m beyond $\widetilde{m}$, too, must contain some non-trivial entries, different from error symbols. But then we also set $\mathsf{win} \leftarrow 1$ in the INT-CST experiment run.

If c contains only deviating (or exceeding) bits or in the case that synchronization was lost before, the full resulting message fragment (m' resp. m) is considered for the winning condition, i.e., whenever $\mathcal{A}$ in this case wins in the INT-PST experiment, it also does in the INT-CST experiment. □

Generic composition. As explained earlier, standard arguments to prove the composition theorem do not apply in the streaming setting. The issue here is that losing the integrity game does not make the output of $\mathcal{O}_{\mathsf{Recv}}$ (in the confidentiality game) predictable. Therefore, any strategy which allows the recovery of the composition theorem should make it possible to forecast the output behavior of the receiving algorithm when certain conditions are met. In line with this observation we introduce a new notion, so-called *error predictability*, which precisely formalizes the ability to efficiently predict (part of) the output of Recv in case error messages are expected. Intuitively speaking, error predictability demands that any error symbols returned by Recv on input the 'next ciphertext' c can be efficiently predicted given only the ciphertext stream C_S output by Send, the ciphertext stream C_R input to Recv, and the ciphertext c.

As formalized in Definition 10.9 and through the security experiment of Figure 10.6, we say that a channel provides *error predictability* (ERR-PRE) with respect to an efficient probabilistic *predictor* algorithm Pred if this predictor Pred, given C_R, C_S, and c, accurately outputs the above-mentioned error string with high probability, for every arbitrarily chosen ciphertext c. Put differently, the ERR-PRE experiment declares its adversary to be successful if it ever queries a (counterfeit) ciphertext c that induces Recv to produce different errors from those output by the predictor. Note that the adversary can always learn if it has won by evaluating the winning condition "$\langle m \rangle_{\mathcal{E}} \neq \mathsf{Pred}(C_S, C_R, c)$" itself for the available data.

Definition 10.9 (Error predictability (ERR-PRE))**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel with error space* $\mathcal{E}$*, and let* Pred *be an efficient probabilistic algorithm. We say that* Ch *provides* error predictability (ERR-PRE) with respect to Pred *if for every PPT adversary* $\mathcal{A}$ *playing in the experiment* ERR-PRE *defined in Figure 10.6 against channel* Ch*, the following advantage function is negligible:*

$$\mathsf{Adv}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{A}}(1^\lambda) = 1\right].$$

We are now ready to formalize the idea that, for the class of error-predictable channels, the generic composition theorem holds (even for channels supporting multiple decryption errors).

Theorem 10.10 (INT-CST $\wedge$ IND-CPFA $\wedge$ ERR-PRE $\Longrightarrow$ IND-CCFA)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a (correct) stream-based channel. If* Ch *provides integrity of ciphertext streams, error predictability with respect to a predictor* Pred*, and indistinguishability under chosen plaintext-fragment attacks then it also provides indistinguishability under chosen ciphertext-fragment attacks. Formally, for every efficient* IND-CCFA *adversary* $\mathcal{A}$ *there exist efficient* INT-CST *adversary* $\mathcal{B}$*,* ERR-PRE *adversary* $\mathcal{C}$*, and* IND-CPFA *adversary* $\mathcal{D}$ *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCFA}}_{\mathsf{Ch},\mathcal{A}} \leq 2 \cdot \mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{B}} + 2 \cdot \mathsf{Adv}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{C}} + \mathsf{Adv}^{\mathsf{IND\text{-}CPFA}}_{\mathsf{Ch},\mathcal{D}}.$$

Proof. We will consider a sequence of game transitions from the IND-CCFA experiment to the IND-CPFA experiment and bound the difference in probability between each game and its successor in the sequence with the advantage of a specific adversary. For better legibility we will denote the intermediate experiments by $\mathsf{E}^{i,b}_{\mathcal{A}}$ for $i \in \{0,1,2\}$ and, with a slight abuse of notation, use the shorthand $\Pr[\mathsf{E}^{i,b}_{\mathcal{A}}]$ to indicate the probability $\Pr[\mathsf{E}^{i,b}_{\mathsf{Ch},\mathcal{A}} = 1]$. In the game transitions we only change the experiment's $\mathcal{O}_{\mathsf{Recv}}$ oracle behavior; the modifications are also shown in Figure 10.7.

Starting from the IND-CCFA experiment of Figure 10.3 against $\mathcal{A}$, that we denote $\mathsf{E}^{0,b}_{\mathcal{A}}$, we define a new experiment $\mathsf{E}^{1,b}_{\mathcal{A}}$ which provides the adversary only with the output of the error predictor in case it breaks ciphertext integrity of streams. More precisely, we modify the $\mathcal{O}_{\mathsf{Recv}}$ oracle in $\mathsf{E}^{1,b}_{\mathcal{A}}$ as follows: we add before Lines 13 and 29 of the original experiment a conditional check for $m \notin \mathcal{E}^*$ (before line 13) resp. $m' \notin \mathcal{E}^*$ (before line 29). If either of these conditions evaluates to true, we set a flag bad^b_I and replace m, resp. m', with the output of $\mathsf{Pred}(C_S, C_R, c)$. Let bad^b_I also denote event that the flag bad^b_I is set to true. Note that $\mathsf{E}^{1,b}_{\mathcal{A}}$ and $\mathsf{E}^{0,b}_{\mathcal{A}}$ execute the same instructions as long as bad^b_I does not happen (beyond bookkeeping of C_R in the $\mathsf{sync} = 0$ case which could also be inserted in $\mathsf{E}^{0,b}_{\mathcal{A}}$ without changing its behavior). We can thus assert that $\Pr[\mathsf{E}^{0,b}_{\mathcal{A}} \wedge \neg bad^b_I] = \Pr[\mathsf{E}^{1,b}_{\mathcal{A}} \wedge \neg bad^b_I]$, and hence obtain the bound $|\Pr[\mathsf{E}^{0,b}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,b}_{\mathcal{A}}]| \leq \Pr[bad^b_I]$ (by the Difference Lemma [Sho06] or the Fundamental Lemma of game playing [BR06]).

We show next how to convert any adversary $\mathcal{A}$ that triggers either event bad^0_I or bad^1_I into an adversary $\mathcal{B}$ that violates the INT-CST security of Ch. Adversary $\mathcal{B}$ initially chooses a bit d uniformly at random and then runs $\mathcal{A}$, answering its queries as follows. If $\mathcal{A}$ queries (m_0, m_1, f) to $\mathcal{O}_{\mathsf{LoR}}$ then $\mathcal{B}$ queries (m_d, f) to its oracle $\mathcal{O}_{\mathsf{Send}}$ and forwards the oracle's answer to $\mathcal{A}$. Similarly $\mathcal{B}$ relays every receiving query c to its oracle $\mathcal{O}_{\mathsf{Recv}}$, obtains a response m, and returns the projection $\langle m \rangle_{\mathcal{E}}$ of m onto the error space $\mathcal{E}$ to $\mathcal{A}$. Note that if the message $\mathcal{A}$ is supposed to obtain were to contain any non-error symbol then it would trigger the bad event. When $\mathcal{A}$ halts, so does $\mathcal{B}$.

As games $\mathsf{E}^{0,b}_{\mathcal{A}}$ and $\mathsf{E}^{1,b}_{\mathcal{A}}$ are the same until bad^b_I we conclude that $\mathcal{B}$ provides a prefect simulation of the games as long as $\mathcal{A}$ does not trigger bad^b_I. Moreover, if $\mathcal{A}$ triggers bad^b_I then $\mathcal{B}$ wins in the INT-CST experiment if it had chosen $d = b$. Thus, we derive the inequality $\mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{B}} \geq \Pr[bad^0_I \wedge d = 0] + \Pr[bad^1_I \wedge d = 1] = \frac{1}{2} \cdot \Pr[bad^0_I] + \frac{1}{2} \cdot \Pr[bad^1_I]$, from which we can bound the advantage of $\mathcal{A}$ in the IND-CCFA experiment as follows:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{IND\text{-}CCFA}}_{\mathsf{Ch},\mathcal{A}} &= \left|\Pr[\mathsf{E}^{0,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{0,0}_{\mathcal{A}}]\right| \\
&\leq \left|\Pr[\mathsf{E}^{0,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,1}_{\mathcal{A}}]\right| + \left|\Pr[\mathsf{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,0}_{\mathcal{A}}]\right| + \left|\Pr[\mathsf{E}^{1,0}_{\mathcal{A}}] - \Pr[\mathsf{E}^{0,0}_{\mathcal{A}}]\right| \\
&\leq \Pr[bad^1_I] + \left|\Pr[\mathsf{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,0}_{\mathcal{A}}]\right| + \Pr[bad^0_I] \\
&\leq 2 \cdot \mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{B}} + \left|\Pr[\mathsf{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,0}_{\mathcal{A}}]\right|.
\end{aligned}$$

Observe that in experiment $\mathsf{E}^{1,b}_{\mathcal{A}}$, if bad^b_I is not triggered, the receiving algorithm when fed with a deviating or exceeding ciphertext fragment either outputs error symbols or an empty

$\mathcal{O}_{\mathsf{Recv}}(c)$ in $\mathsf{E}^{0,b}_{\mathcal{A}}$:	$\mathcal{O}_{\mathsf{Recv}}(c)$ in $\mathsf{E}^{1,b}_{\mathcal{A}}$:	$\mathcal{O}_{\mathsf{Recv}}(c)$ in $\mathsf{E}^{2,b}_{\mathcal{A}}$:
11 if $\mathsf{sync} = 0$ then	1 if $\mathsf{sync} = 0$ then	1 if $\mathsf{sync} = 0$ then
12 $\quad(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$	2 $\quad(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$	2 $\quad(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
	3 $\quad$ if $m \notin \mathcal{E}^*$ then	3 $\quad$ if $m \notin \mathcal{E}^*$ then
	4 $\quad\quad bad^b_I \leftarrow \mathsf{true}$	4 $\quad\quad bad^b_I \leftarrow \mathsf{true}$
	5 $\quad\quad m \xleftarrow{\$} \mathsf{Pred}(C_S, C_R, c)$	5 $\quad\quad m \xleftarrow{\$} \mathsf{Pred}(C_S, C_R, c)$
		6 $\quad e \xleftarrow{\$} \mathsf{Pred}(C_S, C_R, c)$
		7 $\quad$ if $m \neq e$ then
		8 $\quad\quad bad^b_E \leftarrow \mathsf{true}$
	6 $\quad C_R \leftarrow C_R \| c$	9 $\quad C_R \leftarrow C_R \| c$
13 $\quad$ return m	7 $\quad$ return m	10 $\quad$ return e
14 else if $C_R \| c \preccurlyeq C_S$ then	8 else if $C_R \| c \preccurlyeq C_S$ then	11 else if $C_R \| c \preccurlyeq C_S$ then
15 $\quad(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$	9 $\quad(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$	12 $\quad(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
16 $\quad C_R \leftarrow C_R \| c$	10 $\quad C_R \leftarrow C_R \| c$	13 $\quad C_R \leftarrow C_R \| c$
17 $\quad$ return ε	11 $\quad$ return ε	14 $\quad$ return ε
18 else	12 else	15 else
19 $\quad$ if $C_R \prec [C_R \| c, C_S]$ then	13 $\quad$ if $C_R \prec [C_R \| c, C_S]$ then	16 $\quad$ if $C_R \prec [C_R \| c, C_S]$ then
20 $\quad\quad \widetilde{c} \leftarrow [C_R \| c, C_S] \,\%\, C_R$	14 $\quad\quad \widetilde{c} \leftarrow [C_R \| c, C_S] \,\%\, C_R$	17 $\quad\quad \widetilde{c} \leftarrow [C_R \| c, C_S] \,\%\, C_R$
21 $\quad\quad \widetilde{\mathsf{st}_R} \leftarrow \mathsf{st}_R$	15 $\quad\quad \widetilde{\mathsf{st}_R} \leftarrow \mathsf{st}_R$	18 $\quad\quad \widetilde{\mathsf{st}_R} \leftarrow \mathsf{st}_R$
22 $\quad\quad (\widetilde{\mathsf{st}_R}, \widetilde{m}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}_R}, \widetilde{c})$	16 $\quad\quad (\widetilde{\mathsf{st}_R}, \widetilde{m}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}_R}, \widetilde{c})$	19 $\quad\quad (\widetilde{\mathsf{st}_R}, \widetilde{m}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}_R}, \widetilde{c})$
23 $\quad\quad (\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$	17 $\quad\quad (\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$	20 $\quad\quad (\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
24 $\quad\quad m' \leftarrow m \,\%\, [m, \widetilde{m}]$	18 $\quad\quad m' \leftarrow m \,\%\, [m, \widetilde{m}]$	21 $\quad\quad m' \leftarrow m \,\%\, [m, \widetilde{m}]$
25 $\quad$ else	19 $\quad$ else	22 $\quad$ else
26 $\quad\quad (\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$	20 $\quad\quad (\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$	23 $\quad\quad (\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
27 $\quad$ if $C_S \not\preccurlyeq C_R \| c$ or $m' \neq \varepsilon$ then	21 $\quad$ if $C_S \not\preccurlyeq C_R \| c$ or $m' \neq \varepsilon$ then	24 $\quad$ if $C_S \not\preccurlyeq C_R \| c$ or $m' \neq \varepsilon$ then
28 $\quad\quad \mathsf{sync} \leftarrow 0$	22 $\quad\quad \mathsf{sync} \leftarrow 0$	25 $\quad\quad \mathsf{sync} \leftarrow 0$
	23 $\quad$ if $m' \notin \mathcal{E}^*$ then	26 $\quad$ if $m' \notin \mathcal{E}^*$ then
	24 $\quad\quad bad^b_I \leftarrow \mathsf{true}$	27 $\quad\quad bad^b_I \leftarrow \mathsf{true}$
	25 $\quad\quad m' \xleftarrow{\$} \mathsf{Pred}(C_S, C_R, c)$	28 $\quad\quad m' \xleftarrow{\$} \mathsf{Pred}(C_S, C_R, c)$
		29 $\quad e \xleftarrow{\$} \mathsf{Pred}(C_S, C_R, c)$
		30 $\quad$ if $m' \neq e$ then
		31 $\quad\quad bad^b_E \leftarrow \mathsf{true}$
29 $\quad C_R \leftarrow C_R \| c$	26 $\quad C_R \leftarrow C_R \| c$	32 $\quad C_R \leftarrow C_R \| c$
30 $\quad$ return m'	27 $\quad$ return m'	33 $\quad$ return e

Figure 10.7: The modifications in the proof of Theorem 10.10 within the $\mathcal{O}_{\mathsf{Recv}}$ oracle in experiments $\mathsf{E}^{0,b}_{\mathcal{A}}$ (equal to the IND-CCA experiment), $\mathsf{E}^{1,b}_{\mathcal{A}}$, and $\mathsf{E}^{2,b}_{\mathcal{A}}$. Lines in frames in experiment $\mathsf{E}^{i,b}_{\mathcal{A}}$ differ from those in the previous experiment $\mathsf{E}^{i-1,b}_{\mathcal{A}}$. Other lines (on same height) are identical in both experiments. Line numbering in $\mathsf{E}^{0,b}_{\mathcal{A}}$ is as in the IND-CCFA experiment (see Figure 10.3).

string. Using the error predictability of the channel wrt. predictor Pred, we can now predict which one of these two cases actually occurs. To this end, we define a variant of $\mathsf{E}^{1,b}_{\mathcal{A}}$, denoted $\mathsf{E}^{2,b}_{\mathcal{A}}$, in which the receiving oracle additionally processes deviating or exceeding ciphertext fragments using the predictor Pred and provides $\mathcal{A}$ with that output. Furthermore, a flag bad^b_E is set if the output of Pred differs from the output of Recv in these cases. See Figure 10.7 for the precise changes from $\mathsf{E}^{1,b}_{\mathcal{A}}$ to $\mathsf{E}^{2,b}_{\mathcal{A}}$. Let again bad^b_E also denote the event that the flag bad^b_E is set to true. Then $\mathsf{E}^{1,b}_{\mathcal{A}}$ and $\mathsf{E}^{2,b}_{\mathcal{A}}$ execute the same instructions as long as bad^b_E does not happen, and hence their difference in probability is bounded by $\Pr[bad^b_E]$.

Similarly to the previous hop we define an ERR-PRE adversary $\mathcal{C}$ which runs $\mathcal{A}$, chooses a bit d uniformly at random, and simulates games $\mathsf{E}^{i,b}_{\mathcal{A}}$ for $i \in \{1, 2\}$ by relaying $\mathcal{A}$'s queries to its oracles (as above, left-or-right queries (f, m_0, m_1) are first turned into (f, m_d), then sent to $\mathcal{C}$'s oracle). First of all observe that, in the check $m \neq e$ (resp. $m' \neq e$) triggering bad^b_E, it always holds that $m \in \mathcal{E}^*$ (resp. $m' \in \mathcal{E}^*$) and hence $m = \langle m \rangle_{\mathcal{E}}$ resp. $m' = \langle m' \rangle_{\mathcal{E}}$. Thus, if $\mathcal{A}$ triggers bad^b_E, this makes $\mathcal{C}$ win in the ERR-PRE experiment, as $m \neq e$ if and only if

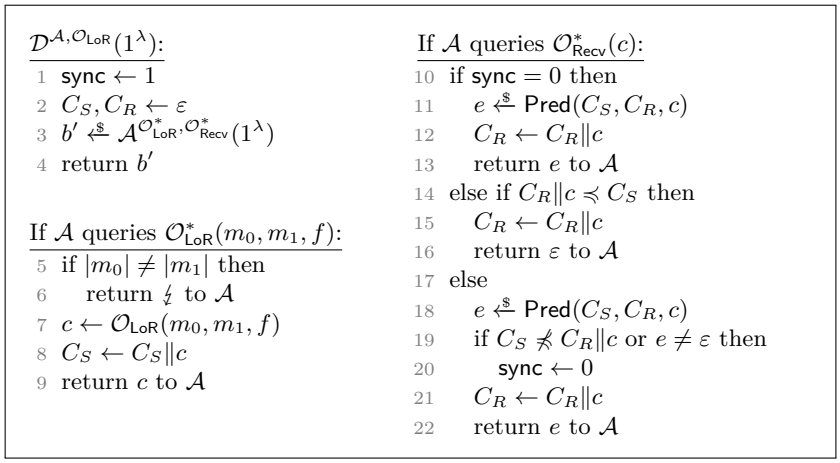


Figure 10.8: IND-CPFA adversary $\mathcal{D}$ simulates the experiment $\mathsf{E}^{2,b}_{\mathcal{A}}$, as in the proof of Theorem 10.10.

$\langle m \rangle_{\mathcal{E}} \neq \mathsf{Pred}(C_S, C_R, c)$ (and likewise for m').

Using a similar argument as above we deduce $\mathsf{Adv}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{C}} \geq \frac{1}{2} \cdot \Pr[bad^0_E] + \frac{1}{2} \cdot \Pr[bad^1_E]$, which allows us to bound the advantage of $\mathcal{A}$ in the second experiment as follows:

$$\begin{aligned}\left|\Pr[\mathsf{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,0}_{\mathcal{A}}]\right| &\leq \left|\Pr[\mathsf{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{2,1}_{\mathcal{A}}]\right| + \left|\Pr[\mathsf{E}^{2,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{2,0}_{\mathcal{A}}]\right| + \left|\Pr[\mathsf{E}^{2,0}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,0}_{\mathcal{A}}]\right| \\ &\leq 2 \cdot \mathsf{Adv}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{C}} + \left|\Pr[\mathsf{E}^{2,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{2,0}_{\mathcal{A}}]\right|.\end{aligned}$$

We finally observe that the indistinguishability game $\mathsf{E}^{2,b}_{\mathcal{A}}$ can be safely emulated using an IND-CPFA adversary $\mathcal{D}$, as shown in Figure 10.8. Here $\mathcal{D}$ is granted oracle access to $\mathcal{O}_{\mathsf{LoR}}$ as in the IND-CPFA experiment of Figure 10.3 and simulates the IND-CCFA oracles $\mathcal{O}^*_{\mathsf{LoR}}$ and $\mathcal{O}^*_{\mathsf{Recv}}$ for $\mathcal{A}$. Adversary $\mathcal{D}$ simply relays $\mathcal{O}^*_{\mathsf{LoR}}$ queries to its oracle $\mathcal{O}_{\mathsf{LoR}}$, while it answers queries to $\mathcal{O}^*_{\mathsf{Recv}}$ on its own by invoking the predictor Pred and returning its output.[53] This leads to the following bound:

$$\left|\Pr[\mathsf{E}^{2,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{2,0}_{\mathcal{A}}]\right| \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPFA}}_{\mathsf{Ch},\mathcal{D}}.$$

Combining the various bounds implied by the above sequence of game transitions yields the stated security bound. □

Remark 10.11 (Error predictability vs. error simulatability). After the original publication of this work [FGMP15], Barwell et al. introduced the notion of *error simulatability* for subtle authenticated encryption [BPS15]. Error simulatability is similar in spirit to, but seemingly weaker than, error predictability (the latter requires the existence of an efficient algorithm that outputs *the same* errors produced by Recv while the former only demands that simulated errors be *indistinguishable*[54] from those output by Recv). In fact, since error predictability is defined for a stateful primitive and error simulatability accounts for a stateless primitive, the two notions are incomparable. However, if error simulatability were to be adapted to the streaming setting, it seems plausible that Theorem 10.10 also holds under such weaker requirement. We leave open

[53] We note that $\mathcal{D}$ could actually always downright invoke the error predictor, which would be conceptually closer to the original composition theorem proof for stateful encryption [BKN04]. We however decided to follow the in-sync/out-of-sync case distinction to facilitate comparison with the structure of $\mathcal{O}_{\mathsf{Recv}}$ in $\mathsf{E}^{2,b}_{\mathcal{A}}$.

[54] More precisely, a *subtle AE* scheme is an AE scheme augmented with a 'leakage function' describing how (a specific implementation of) the decryption algorithm reacts when processing invalid ciphertexts. Thus, according to [BPS15], the simulator's output need not be indistinguishable from the output of the decryption algorithm, but rather from the output of the leakage function.

how to adapt error simulatability to the context of stream-based channels and to investigate further how it relates to error predictability.

10.4 Generic Construction of Stream-Based Channels from AEAD

In this section we demonstrate the feasibility of our security notions by providing a generic construction of stream-based channels which is directly based on the well-established primitive of authenticated encryption with associated data (AEAD, cf. Section 9.2) and provides strong security in terms of confidentiality as well as integrity. Although being illustrative rather than definitive, we remark that our construction is quite close to the TLS record protocol.

10.4.1 The Construction

We propose a generic construction of a stream-based channel $\mathsf{Ch}_{\mathsf{AEAD}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ from any (randomized) AEAD scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$ and error symbol $\perp$. We assume that AEAD supports encryption of variable-length messages of up to il bits and that the ciphertext output for any such message has bit-length at most $2^{\mathsf{ol}} - 1$, for $\mathsf{il}, \mathsf{ol} \in \mathbb{N}$. This enables us to encode the length of each AEAD ciphertext with a fixed-size string of ol bits.

Our channel construction $\mathsf{Ch}_{\mathsf{AEAD}}$ has sending state space $\mathcal{S}_S = \mathcal{K} \times \mathbb{N} \times \{0,1\}^*$, receiving state space $\mathcal{S}_R = \mathcal{K} \times \mathbb{N} \times \{0,1\}^* \times \{0,1\}$, and error space $\mathcal{E} = \{\perp\}$. In Figure 10.9 we give an algorithmic specification of our construction, and describe it next in detail.

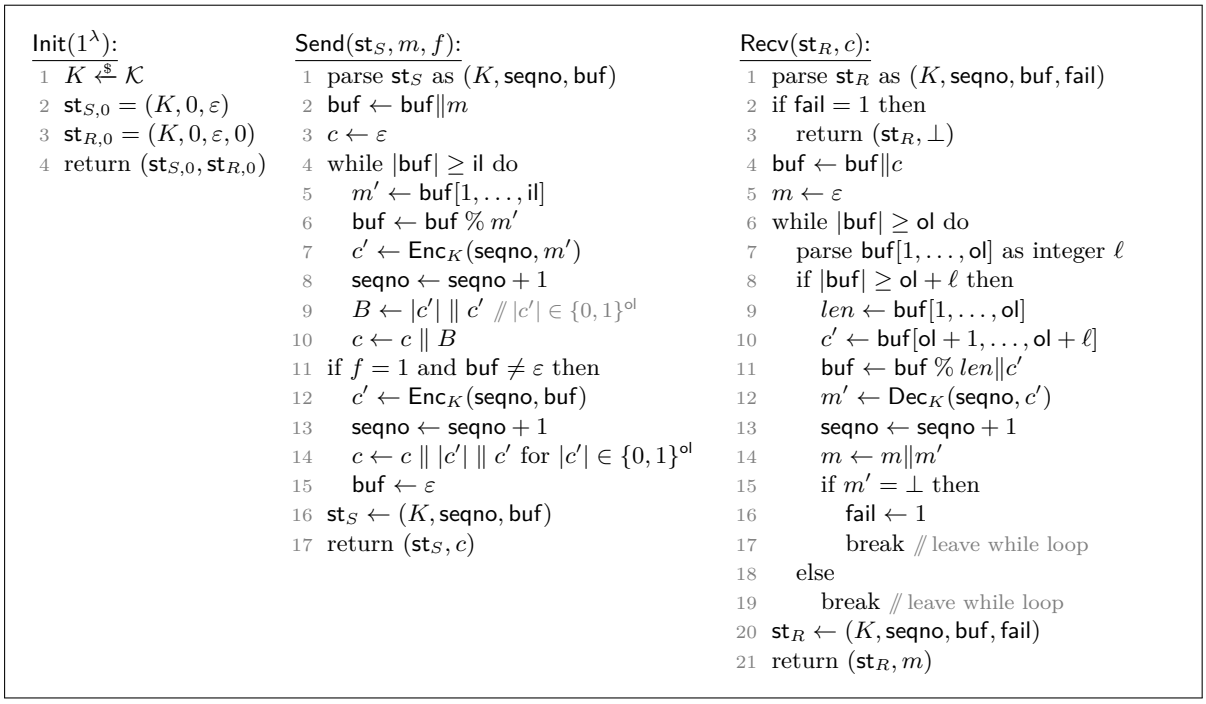
Construction 10.12 (AEAD-based construction $\mathsf{Ch}_{\mathsf{AEAD}}$)**.** *Consider an AEAD scheme* $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ *with key space* $\mathcal{K}$ *and error symbol* $\perp$*. We define* $\mathsf{Ch}_{\mathsf{AEAD}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *to be the stream-based channel algorithmically specified in Figure 10.9 and described in detail below.*

- The Init algorithm first draws uniformly at random a key $K \xleftarrow{\$} \mathcal{K}$ for the AEAD scheme. It then initializes the sending and receiving state respectively as tuples containing key K, a sequence number set to 0, and a message-fragment resp. ciphertext-fragment buffer initially empty; the receiving state also contains a failure flag fail, initially set to 0.

- The Send algorithm keeps on buffering input message strings until it has collected at least il bits. If sufficiently many bits have been collected, then Send invokes the AEAD encryption algorithm on input message chunks m' of length $|m'| = \mathsf{il}$ and a running sequence number seqno as associated data.[55] The corresponding AEAD ciphertext c' is then prepended with the binary encoding of its size, i.e., the bitstring $len = |c'|$, and the resulting string is appended to the ciphertext string c to be output. In case the Send algorithm was called with the flush flag set to 1, in a final step it also AEAD encrypts any remaining buffered message in the same way, in order to empty the message buffer (this message will potentially contain less than il bits). Note that the size encoding len is a bitstring of fixed length ol and it is not authenticated. Henceforth we may refer to the concatenation of an AEAD ciphertext c' prepended with its size encoding len as a 'block' $B = len \| c'$ (see line 9 in Figure 10.9).

- The Recv algorithm outputs an error (without any further state modification) once a first error has emerged from the AEAD decryption algorithm in some previous call (this is

[55] With a nonce-based AEAD scheme one could use seqno as the encryption nonce and have empty associated data input, as done, e.g., in the TLS 1.3 record protocol from draft-11 [Res15f] on (cf. our construction 12.8 in Chapter 12). We have chosen the present construction because of its closeness to TLS 1.2 [DR08] and initial drafts of TLS 1.3 (up to draft-10 [Res15e]), which treat the sequence number as associated data.



```
Init(1^λ):
1  K ←$ 𝒦
2  st_{S,0} = (K, 0, ε)
3  st_{R,0} = (K, 0, ε, 0)
4  return (st_{S,0}, st_{R,0})

Send(st_S, m, f):
1  parse st_S as (K, seqno, buf)
2  buf ← buf‖m
3  c ← ε
4  while |buf| ≥ il do
5     m' ← buf[1, ..., il]
6     buf ← buf % m'
7     c' ← Enc_K(seqno, m')
8     seqno ← seqno + 1
9     B ← |c'| ‖ c'  // |c'| ∈ {0,1}^ol
10    c ← c ‖ B
11 if f = 1 and buf ≠ ε then
12    c' ← Enc_K(seqno, buf)
13    seqno ← seqno + 1
14    c ← c ‖ |c'| ‖ c' for |c'| ∈ {0,1}^ol
15    buf ← ε
16 st_S ← (K, seqno, buf)
17 return (st_S, c)

Recv(st_R, c):
1  parse st_R as (K, seqno, buf, fail)
2  if fail = 1 then
3     return (st_R, ⊥)
4  buf ← buf‖c
5  m ← ε
6  while |buf| ≥ ol do
7     parse buf[1, ..., ol] as integer ℓ
8     if |buf| ≥ ol + ℓ then
9        len ← buf[1, ..., ol]
10       c' ← buf[ol + 1, ..., ol + ℓ]
11       buf ← buf % len‖c'
12       m' ← Dec_K(seqno, c')
13       seqno ← seqno + 1
14       m ← m‖m'
15       if m' = ⊥ then
16          fail ← 1
17          break  // leave while loop
18    else
19       break  // leave while loop
20 st_R ← (K, seqno, buf, fail)
21 return (st_R, m)
```

Figure 10.9: Generic construction of a stream-based channel $\mathsf{Ch}_{\mathsf{AEAD}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ from any authenticated encryption with associated data (AEAD) scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$ and distinguished error symbol $\perp$ which allows to encrypt variable-length messages of up to il bits and for which the ciphertext output has length at most $2^{\mathsf{ol}} - 1$ bits.

indicated by the failure flag set to $\mathsf{fail} = 1$); otherwise, it appends the incoming ciphertext fragment to its buffer. In case enough bits to parse the length field of ol bits were received it does so. Next, it checks whether the buffer contains a complete AEAD ciphertext c' of the indicated length len and, if so, strips it from the buffer, decrypts it (incrementing the sequence number used in the associated data), and appends the result m' to the message m to be output. This process is repeated until there is no full block $B = len \| c'$ left in the buffer. However, in case the AEAD decryption algorithm outputs an error, after appending the error symbol $\perp$ to the output message, the Recv algorithm sets the failure flag to 1 and stops parsing further input.

Correctness. For correctness, observe that the Send algorithm generates a ciphertext output which always consists of blocks of ol bits plus the number ℓ of bits binary encoded in these ol bits (at most $2^{\mathsf{ol}} - 1$), where the sequence number seqno is increased with each such block output. Moreover, when called with the flush flag set to 1, the Send algorithm ensures that the entire message stream input so far is written into the ciphertext being output. The Recv algorithm buffers its input and re-establishes the same blocks of $\mathsf{ol} + \ell$ bits as generated by Send, thus also assigning the same sequence number to each block. Since the AEAD decryption algorithm in Recv is called on the same sequence number and exactly the output generated by the encryption algorithm call in Send, we obtain correctness for our generic stream-based channel construction $\mathsf{Ch}_{\mathsf{AEAD}}$ by virtue of the correctness of the AEAD scheme.

10.4.2 Security Analysis

Our generic stream-based channel construction $\mathsf{Ch}_{\mathsf{AEAD}}$ from Construction 10.12 provides indistinguishability under chosen plaintext-fragment attacks (IND-CPFA), integrity of ciphertext streams (INT-CST), and error predictability (ERR-PRE), given that the underlying authenticated encryption with associated data scheme AEAD provides indistinguishability under chosen plaintext attacks (IND-CPA) and integrity of ciphertexts (INT-CTXT) as defined in Section 9.2. Using Theorem 10.10 we can moreover infer that it also provides indistinguishability under chosen ciphertext-fragment attacks (IND-CCFA).

Theorem 10.13 (IND-CPFA security of $\mathsf{Ch}_{\mathsf{AEAD}}$)**.** *The stream-based channel* $\mathsf{Ch}_{\mathsf{AEAD}}$ *from Construction 10.12 provides indistinguishability under chosen plaintext-fragment attacks (*IND-CPFA*) if the authenticated encryption with associated data scheme* AEAD *provides indistinguishability under chosen plaintext attacks (*IND-CPA*). Formally, for every efficient* IND-CPFA *adversary* $\mathcal{A}$ *against* $\mathsf{Ch}_{\mathsf{AEAD}}$ *there exists an efficient* IND-CPA *adversary* $\mathcal{B}$ *against* AEAD *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPFA}}_{\mathsf{Ch}_{\mathsf{AEAD}},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AEAD},\mathcal{B}}(\lambda).$$

Proof. We reduce the IND-CPFA security of $\mathsf{Ch}_{\mathsf{AEAD}}$ to the IND-CPA security of AEAD by constructing from an efficient adversary $\mathcal{A}$ against the former property an efficient adversary $\mathcal{B}$ against the latter property. In order to simulate the $\mathcal{O}_{\mathsf{LoR}}$ oracle for $\mathcal{A}$ we let $\mathcal{B}$ perform the buffering and sending procedure as defined for Send in Figure 10.9, keeping two buffers for the two message inputs m_0 and m_1 from $\mathcal{A}$ and a sequence number. As the buffering behavior and sending procedure only depends on the length of the input message to Send but not its content, the message blocks to be encrypted using the AEAD scheme are treated identically for either the m_0 or the m_1 buffer. This allows $\mathcal{B}$ to replace the encryption operations by calls to its encryption oracle in the IND-CPA game with the according blocks (of same size) both from the m_0 and m_1 buffer and the (always coinciding) sequence number. This results in a single ciphertext which $\mathcal{B}$ can then process further, as defined in the Send algorithm, to provide the output ciphertext to $\mathcal{A}$. Finally, when $\mathcal{A}$ outputs its guessed bit b', we let $\mathcal{B}$ output the same bit as its guess.

Assume $\mathcal{A}$ is successful. Since $\mathcal{B}$ perfectly simulates the $\mathcal{O}_{\mathsf{LoR}}$ oracle for $\mathcal{A}$, it also wins in the IND-CPFA game. □

Theorem 10.14 (INT-CST security of $\mathsf{Ch}_{\mathsf{AEAD}}$)**.** *The stream-based channel* $\mathsf{Ch}_{\mathsf{AEAD}}$ *from Construction 10.12 provides integrity of ciphertext streams (*INT-CST*) if the authenticated encryption with associated data scheme* AEAD *provides integrity of ciphertexts (*INT-CTXT*). Formally, for every efficient* INT-CST *adversary* $\mathcal{A}$ *against* $\mathsf{Ch}_{\mathsf{AEAD}}$ *there exists an efficient* INT-CTXT *adversary* $\mathcal{B}$ *against* AEAD *such that*

$$\mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch}_{\mathsf{AEAD}},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD},\mathcal{B}}(\lambda).$$

Proof. Recall that the receiving algorithm Recv of our channel construction processes the ciphertext stream by identifying blocks $B_1, B_2, \ldots$ with $B_i = len_i \| c'_i$ where c'_i is an AEAD ciphertext and len_i is the (fixed-length) binary encoding of its size $|c'_i|$. The message fragments output by Recv are obtained by concatenating the AEAD decryptions m'_i of the so identified ciphertexts c'_i. In particular, Recv produces some non-trivial output if and only if it processes at least a *full block* B_i. The main observation is that, in order to break the INT-CST property of $\mathsf{Ch}_{\mathsf{AEAD}}$, an adversary must submit to $\mathcal{O}_{\mathsf{Recv}}$ a non-genuine (i.e., deviating or exceeding) ciphertext stream whose non-genuine part contains a full valid block $B^* = len^* \| c'^*$. More precisely, the AEAD decryption of c'^* with the current sequence number seqno as associated data must yield some valid message m^*. Now, since the scheme increases seqno before each

AEAD encryption, no associated data is ever repeated. Moreover, by hypothesis the block B^* deviates from the genuine ciphertext stream. Thus (seqno, c'^*) is an AEAD forgery.

We now formalize this intuition. Let $\mathcal{A}$ be an adversary attacking the INT-CST of channel $\mathsf{Ch}_{\mathsf{AEAD}}$. We build an adversary $\mathcal{B}$ which runs $\mathcal{A}$ internally as a black box and breaks the INT-CTXT property of AEAD as long as $\mathcal{A}$ is successful against the INT-CST property of $\mathsf{Ch}_{\mathsf{AEAD}}$. Adversary $\mathcal{B}$ emulates the channel construction (see Figure 10.9) by forwarding AEAD encryptions to the oracle $\mathcal{O}_{\mathsf{Enc}}(\cdot,\cdot)$ provided in the INT-CTXT security experiment and by performing the buffering steps on its own. For this it keeps buffer strings buf_S, buf_R and a sequence number seqno, initialized to the empty strings and to zero respectively. It also keeps lists $\mathbf{m}'$ and $\mathbf{c}'$ for bookkeeping of sent AEAD messages and ciphertexts respectively, as well as a string C_R in which it registers the received (stream) ciphertext fragments.

$\mathcal{B}$ answers $\mathcal{A}$'s queries as follows.

- When $\mathcal{A}$ poses a *sending query* (m, f), $\mathcal{B}$ appends m to the buffer buf_S, initializes an empty ciphertext c, and repeats the steps of instructions 4–10 from Figure 10.9. In particular, $\mathcal{B}$ performs the encryption steps by querying $\mathcal{O}_{\mathsf{Enc}}$ on tuples (ad, m') where $ad = \mathsf{seqno}$ is a running sequence number, and registers the AEAD messages m' and ciphertexts c' in the lists $\mathbf{m}'$ and $\mathbf{c}'$, respectively, according to the order in which they are processed. If the flush flag is set to $f = 1$, $\mathcal{B}$ executes one more encryption (oracle call) using as m' the remaining buffer. Finally it returns c to $\mathcal{A}$.

- When $\mathcal{A}$ poses a *receiving query* c, the reduction updates buffer buf_R and string C_R by appending c to both of them and checks if c is genuine by comparing the components of $\mathbf{c}'$ with the blocks $B_i = len_i \| c'_i$ contained in C_R.

  If c is genuine, $\mathcal{B}$ traverses the buffer buf_R and identifies the blocks B_i corresponding to the sent AEAD ciphertexts. Recall that for each block $B_i = len_i \| c'_i$ the AEAD ciphertext c'_i is registered in the list $\mathbf{c}'$ and, correspondingly, its decryption m' is registered in the list $\mathbf{m}'$. Thus, $\mathcal{B}$ can for each identified ciphertext c'_i recover the decryption m'_i. After this, $\mathcal{B}$ removes the identified blocks $B_i = len_i \| c'_i$ from the buffer buf_R and concatenates all corresponding messages m'_i in the same order they appear in $\mathbf{m}'$, obtaining a string m; finally $\mathcal{B}$ gives m to $\mathcal{A}$.

  If c is not genuine, $\mathcal{B}$ first performs the procedure above, but using instead of c its longest genuine prefix $\widetilde{C}$ *that contains only full blocks* B_i. Note that after this step the buffer buf_R will be empty (because it only contained full blocks and all these are processed). Afterwards $\mathcal{B}$ tries to extract a forgery from the remaining part of c and potentially subsequent queries: if $c \,\%\, \widetilde{C}$ contains a full block B^* then $\mathcal{B}$ immediately extracts a forgery, otherwise it keeps answering with the empty string all subsequent receiving queries until it gets enough ciphertext bits to extract a forgery. As soon as the buffer C_R is augmented with at least a non-genuine full block $B^* = len^* \| c'^*$, the reduction outputs as forgery (ad^*, c'^*) with $ad^* = \mathsf{seqno} + 1$ and halts. Note that $\mathsf{Ch}_{\mathsf{AEAD}}$ by construction from the first occurring AEAD error on only outputs outputs errors. Hence, if $\mathcal{A}$ succeeds at all in breaking integrity, then it will be with the first deviating ciphertext, which consequently $\mathcal{B}$ outputs as its forgery attempt.

It is immediate to see that $\mathcal{B}$ performs a sound simulation of the INT-CST experiment. Indeed, for answering sending queries it executes the same instructions as Send (only the AEAD encryption is replaced with an oracle call to $\mathcal{O}_{\mathsf{Enc}}$, however, the AEAD encryption takes place within the oracle). Although $\mathcal{B}$ lacks a decryption oracle and, thus, cannot process $\mathcal{A}$'s receiving queries, it can answer all genuine queries since, for these, the AEAD decryption is correct. In the same way $\mathcal{B}$ can recover the longest genuine message fragment underlying the first non-genuine

query. Regarding non-genuine decryption queries, $\mathcal{B}$ can either extract an AEAD forgery, or trivially answer by returning an empty message and waiting for more ciphertext bits.

It remains to show that if $\mathcal{A}$ breaks the INT-CST property of $\mathsf{Ch}_{\mathsf{AEAD}}$ then $\mathcal{B}$ is successful in the INT-CTXT game against AEAD. Let c denote $\mathcal{A}$'s first out-of-sync query, let $\widetilde{c}$ be the longest in-sync prefix of c, and let m and $\widetilde{m}$ be the message fragments that Recv would output on input c and $\widetilde{c}$ respectively in the real execution of the INT-CST experiment.

Assume first that $\mathcal{A}$ is successful with its first non-genuine query to $\mathcal{O}_{\mathsf{Recv}}$: we thus have $m \,\%\, [m, \widetilde{m}] \notin \mathcal{E}^*$. Suppose that the (genuine) ciphertext stream that Recv would process up to $\widetilde{c}$ contains the first i AEAD blocks $B_1, \ldots, B_i$ sent. By construction $\widetilde{m} = m'_1 \| \cdots \| m'_i$ where each m'_i is the AEAD decryption of c'_i. Then the ciphertext fragment $c \,\%\, (B_1 \| \cdots \| B_i)$ contains as a prefix a full block $B^* = len^* \| c'^*$ such that c'^* with associated data $\mathsf{seqno}^* = i + 1$ AEAD-decrypts to $m^* \neq \bot$, otherwise we would have either $m \,\%\, [m, \widetilde{m}] \in \mathcal{E}$ or $m \,\%\, [m, \widetilde{m}] = \varepsilon$, against our hypothesis.

The argument above easily extends to the general case in which $\mathcal{A}$ poses several non-genuine queries to $\mathcal{O}_{\mathsf{Recv}}$ before breaking the INT-CST security of $\mathsf{Ch}_{\mathsf{AEAD}}$ by letting $\mathcal{O}_{\mathsf{Recv}}$ create some non-trivial output: $\mathcal{B}$ simply keeps buffering and answering queries with an empty string until it collects enough ciphertext bits to form a full block $B^* = len^* \| c'^*$ (here 'enough' means $\mathsf{ol} + len^*$ bits).

In both cases, once $\mathcal{B}$ obtains sufficiently many ciphertext bits to isolate a block B^*, it stops the simulation and returns as valid AEAD forgery the pair (seqno^*, c'^*). □

Theorem 10.15 (ERR-PRE security of $\mathsf{Ch}_{\mathsf{AEAD}}$)**.** *The stream-based channel* $\mathsf{Ch}_{\mathsf{AEAD}}$ *from Construction 10.12 provides error predictability (*ERR-PRE*), with respect to the predictor* Pred *given in the proof of the theorem, if the authenticated encryption with associated data scheme* AEAD *provides integrity of ciphertexts (*INT-CTXT*). Formally, for every efficient* ERR-PRE *adversary* $\mathcal{A}$ *against* $\mathsf{Ch}_{\mathsf{AEAD}}$ *and predictor* Pred *there exists an efficient* INT-CTXT *adversary* $\mathcal{B}$ *against* AEAD *such that*

$$\mathsf{Adv}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch}_{\mathsf{AEAD}},\mathsf{Pred},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD},\mathcal{B}}(\lambda).$$

Proof. We start with defining the predictor algorithm Pred. On input $C_S \in \{0,1\}^*$, $C_R \in \{0,1\}^*$, and $c \in \{0,1\}^*$ the predictor first computes $C'_R \in \{0,1\}^*$ by removing from $C_R \| c$ the longest prefix with C_S which only consists of complete blocks containing each a length field (of ol bits, where ol is determined by the AEAD scheme) followed by as many bits as binary encoded in that length field. In case C'_R contains a complete block (length field plus encoded number of bits), Pred outputs the distinguished error symbol $\bot$ of the AEAD scheme, otherwise it outputs the empty string ε.

Now we reduce the error predictability ERR-PRE of $\mathsf{Ch}_{\mathsf{AEAD}}$ to the INT-CTXT security of AEAD by turning any efficient adversary $\mathcal{A}$ that distinguishes the output of Recv from the output of Pred into an efficient adversary $\mathcal{B}$ against the INT-CTXT property of the AEAD scheme. Initially, $\mathcal{B}$ sets $C_S \leftarrow \varepsilon$, $C_R \leftarrow \varepsilon$. As in the previous two proofs, it simulates the oracle $\mathcal{O}_{\mathsf{Send}}$ for $\mathcal{A}$ using the encryption oracle in the INT-CTXT game and furthermore appends the obtained result c to C_S.

For simulating the $\mathcal{O}_{\mathsf{Recv}}$ oracle for $\mathcal{A}$, adversary $\mathcal{B}$ appends c to C_R and returns the empty string ε to $\mathcal{A}$ as long as the ciphertext fragments provided are in sync ($C_R \| c \preccurlyeq C_S$). When $\mathcal{A}$ provides ciphertext fragments such that the receiving buffer at some point contains a full block (consisting of an encoded length and the ciphertext of that length) which at some point deviates from the genuine ciphertext stream (i.e., was never output by $\mathcal{B}$'s simulation of $\mathcal{O}_{\mathsf{Send}}$), $\mathcal{B}$ outputs the ciphertext of this block along with the current sequence number value as associated data field as its forgery in the INT-CTXT game and stops.

Note that $\mathcal{B}$ perfectly simulates the ERR-PRE experiment for $\mathcal{A}$, as by correctness of $\mathsf{Ch}_{\mathsf{AEAD}}$ and the buffering behavior of Recv, the first point where Recv could output an error symbol

is when it received a complete ciphertext block which deviates from the ciphertext stream generated by Send. Up to this point, also Pred will not have output an error, so that $\mathcal{A}$ cannot have won yet.

Assume $\mathcal{A}$ wins at the point where the input ciphertext c completes the first deviating ciphertext block input to Recv. As this requires that $\langle m \rangle_{\mathcal{E}} \neq \mathsf{Pred}(C_S, C_R, c)$ but the output of Pred will be $\perp$, this means that the ciphertext block (and the according sequence number as associated data) decrypts under the AEAD scheme to a valid message (and not the distinct error symbol). Hence, this output constitutes a valid forgery and $\mathcal{B}$ thus also wins in the INT-CTXT game. Furthermore, note that $\mathcal{A}$ cannot win in the ERR-PRE experiment with a later call to its $\mathcal{O}_{\mathsf{Recv}}$ oracle, because after the first error occurred, both Recv and Pred consistently output $\perp$ for any further ciphertext fragment input c. □

Applying Theorem 10.10 we can now deduce the following corollary from Theorems 10.13–10.15.

Corollary 10.16 (IND-CCFA security of $\mathsf{Ch}_{\mathsf{AEAD}}$)**.** *The stream-based channel* $\mathsf{Ch}_{\mathsf{AEAD}}$ *from Construction 10.12 provides indistinguishability under chosen ciphertext-fragment attacks (*IND-CCFA*) if the authenticated encryption with associated data scheme* AEAD *provides indistinguishability under chosen plaintext attacks (*IND-CPA*) and integrity of ciphertexts (*INT-CTXT*).*

10.4.3 A Note on the TLS Record Protocol

As discussed earlier, the Transport Layer Security (TLS) record protocol implements a stream-based channel whose complete analysis as such lies outside of the scope of this work. However we do pause to note that our Construction 10.12 of a stream-based channel based on authenticated encryption with associated data is conceptually close to the TLS record protocol when using an AEAD scheme as specified for TLS version 1.2 [DR08, Section 6.2.3.3] and in the current draft for TLS version 1.3 [Res18, Section 5]: the record protocol also incorporates a sequence number which is authenticated but not sent on the wire, and a length field which is sent and authenticated in TLS 1.2 (and which is sent but not authenticated in the draft TLS 1.3).[56] However, the TLS record protocol in version 1.2 additionally includes a 2-byte version number and a 1-byte content type; these are both sent and authenticated in the associated data. Moreover, the AEAD schemes used are considered to be nonce-based, with the TLS 1.3 draft specifying how the nonce is formed and TLS 1.2 leaving the exact nonce generation to be specified by the particular cipher suite in use. TLS (in both versions) furthermore specifies padding mechanisms and TLS 1.3 uses a double-header structure for backwards compatibility reasons.

The content type field in particular allows TLS to multiplex data streams for different purposes within a single connection stream, as TLS 1.2 does for the Handshake Protocol, the Alert Protocol, the ChangeCipherSpec protocol, and the Application protocol. While our model does not capture multiplexing several message streams into one ciphertext stream, it can be augmented to do so. This brings additional complexity and is an avenue for future work.

TLS 1.3 moreover permits updating the encryption key while maintaining the channel's operation, a feature that we treat with our model for multi-key channels in Chapter 12.

[56] That is, our approach of using a length field which is sent on the wire but not part of the authenticated associated data of the AEAD ciphertext conforms with the approach adopted in the TLS 1.3 draft. In contrast to our approach, the TLS 1.3 draft implicitly authenticates the sequence number by letting it enter the AEAD nonce rather than explicitly authenticating it in the associated data field.

Chapter 11

Atomic-Message Channels Supporting Fragmentation

Summary. In this chapter we study the question how applications can safely send atomic messages over a network with possibly fragmented transport. We present our notion of atomic-message channels supporting fragmentation treating such scenarios and define corresponding security notions for confidentiality and integrity. We then develop a generic "encode-then-stream" paradigm for building such channels on top of a stream-based channel (cf. Chapter 10). This construction closely mimics the approach of message-based applications in practice to encode boundaries of messages when running over a stream-based channel, also shedding some light on attacks leveraging ciphertext fragmentation. The results in this chapter are based on the extended full version [FGMP17] of a work published at CRYPTO 2015 [FGMP15].

11.1 Introduction

Many application layer protocols rely on a stream-based transport protocol like TCP, as the data they transmit is inherently *stream-based* (like in audio or video streaming applications) or may consist of individual messages that are too large to be written to, or read from, the transport protocol in one go (as in HTTP [FGM+97] transfers of large files). For such applications, our security model for stream-based channels established in Chapter 10 appropriately captures security guarantees to be expected from a cryptographic stream-based transport protocol or channel such as TLS [DR08, Res18], SSH [YL06a], or QUIC [QUI].

Other application layer protocols, however, are inherently *message-based* (e.g., chat protocols or header transmission in HTTP) and might crucially demand that only messages that are guaranteed to be authentic and complete (i.e., non-truncated) are delivered. Most secure transport protocols in use (e.g., TLS), though, are stream-based in nature and might deliver input messages—which they consider as *fragments*—in several parts, both in real-world implementations and in our model for stream-based channels. Inherently, the streaming nature of the data transmitted and its fragmentation in a stream-based channel (e.g., into chunks of at most 2^{14} bytes in the TLS record protocol) yields a different and intuitively weaker notion of integrity (see also Remark 10.7 on page 158): integrity in the streaming setting does not attach any importance to the boundaries of message (or ciphertext) fragments. Unwary processing of message fragments on the receiver's side might thus break, and has broken in the past, the security of message-dependent application layer protocols (see, e.g., [SP13, BDF+14]). This raises the following safety question: How can a reliable and secure transport channel for *atomic messages* be provided on top of a secure *stream-based* channel in order to protect

message-dependent application protocols from misinterpreting *partial* messages on the receiver's side as *complete* ones?

Note that previous works on channels do not provide a satisfying solution to this problem. For instance, while Bellare et al. [BKN04] and follow-up works (e.g., [KPB03, PRS11, JKSS12, BDPS14, BSWW13]) consider the transport of atomic messages, their model lacks potential fragmentation on the network. Boldyreva et al. [BDPS12] made a first step further in this direction by considering confidentiality of channels that treat messages atomically at the sender's side and allow for fragmentation on the network. Integrity in this setting was later, and concurrently to our work, defined by Albrecht et al. [ADHP16]. Still, while their notions approach capturing atomic-message channels over fragmented transport, they do not provide an answer to the question of how to achieve such from a given stream-based channel.

Atomic-message channels supporting fragmentation. To fill this gap we first introduce the notion of *atomic-message channels supporting fragmentation* which covers schemes that transport atomic messages in a secure way over a potentially fragmenting network. While our (strong) confidentiality and integrity notions are similar in spirit to those defined by Boldyreva et al. [BDPS12] and Albrecht et al. [ADHP16], our syntax already intrinsically encodes an atomic-message interface both for the input on the sender's side as well as for the output on the receiver's side. Beyond that, we further define the corresponding weaker variants of chosen-plaintext confidentiality and plaintext integrity for atomic-message channels supporting fragmentation.

With a rigorous security goal in mind we can confirm that the straightforward approach (used, e.g., in HTTP) to encode distinguished end-of-message symbol into the message stream, thus allowing the receiver to reconstruct the message boundaries, does achieve the desired security goal. After looking closely at this approach we develop a generic paradigm, that we call *encode-then-stream*, for building atomic-message channels that safely transport atomic messages over a stream-based channels, and study its security. Our strategy is to add an encoding layer that turns a message sequence into a stream of bits and allows to recover from that bit stream the original message sequence; then, we simply transmit the obtained bit stream through any secure stream-based channel. The resulting construction provably achieves strong confidentiality and integrity guarantees, as we show, provided that the underlying stream-based channel also offers strong security.

Ultimately, the study of atomic-message security in the presence of ciphertext fragmentation casts a formal light on the truncation [SP13] and 'cookie-cutter' [BDF+14] attacks on HTTP running over TLS, showing how they can be seen as arising from a misunderstanding of the security guarantees that can be provided by a stream-based channel to applications expecting an atomic-message channel. In essence, applications relying on non–integrity-protected end-of-message indicators cannot hope to safely reconstruct atomic messages on the receiving end of a (stream-based) channel.

11.2 Syntax and Functionality of Atomic-Message Channels Supporting Fragmentation

The syntax of atomic-message channels supporting fragmentation for the sending algorithm aSend reverts back to the classical setting with an atomic message input and an atomic ciphertext output, obviating the need for a flush flag as seen for stream-based channels (cf. Section 10.2). Note also that we therefore use the vector notation where appropriate, e.g., $\mathbf{m}[1,\ldots,i]$ then refers to the first i entries of the vector $\mathbf{m}$ (and not the first i bits of some string m).

To capture ciphertext fragmentation on the underlying network we allow the receiving algorithm aRecv to take a ciphertext fragment as input, but we syntactically require it to output clearly separated atomic messages (instead of chunks of a message stream where the boundaries of individual chunks have no inherent meaning, as for stream-based channels). As a ciphertext fragment might contain more than one original ciphertext output by aSend, we need to allow aRecv to output not only a single message but a vector of messages (which may be empty). In contrast to stream-based channels here we consider a generic message space $\mathcal{M} \subseteq \{0,1\}^*$ instead of fixing $\mathcal{M} = \{0,1\}^*$.

Definition 11.1 (Syntax of atomic-message channels supporting fragmentation)**.** *An* atomic-message channel supporting fragmentation $\mathsf{aCh} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *with associated message space $\mathcal{M}$, sending and receiving state space $\mathcal{S}_S$ resp. $\mathcal{S}_R$, and error space $\mathcal{E}$, where $\mathcal{E} \cap \mathcal{M} = \emptyset$, consists of three efficient algorithms:*

- $\mathsf{aInit}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_{S,0}, \mathsf{st}_{R,0})$. *On input a security parameter 1^λ, this probabilistic algorithm outputs initial states $\mathsf{st}_{S,0} \in \mathcal{S}_S$, $\mathsf{st}_{R,0} \in \mathcal{S}_R$ for the sender and the receiver, respectively.*

- $\mathsf{aSend}(\mathsf{st}_S, m) \xrightarrow{\$} (\mathsf{st}'_S, c)$. *On input a sending state $\mathsf{st}_S \in \mathcal{S}_S$ and a message $m \in \mathcal{M}$, this (possibly) probabilistic algorithm outputs an updated state $\mathsf{st}'_S \in \mathcal{S}_S$ and a ciphertext $c \in \{0,1\}^*$.*

- $\mathsf{aRecv}(\mathsf{st}_R, c) \to (\mathsf{st}'_R, (m_1, \ldots, m_\ell))$. *On input a receiving state $\mathsf{st}_R \in \mathcal{S}_R$ and a ciphertext fragment $c \in \{0,1\}^*$, this deterministic algorithm outputs an updated state $\mathsf{st}'_R \in \mathcal{S}_R$ and a (potentially empty) vector of messages (or error symbols) $(m_1, \ldots, m_\ell) \in (\mathcal{M} \cup \mathcal{E})^*$.*

We also use the shorthand *atomic-message channels* to indicate atomic-message channels supporting fragmentation. Furthermore, we use the same vector notation as for stream-based channels from Section 10.2 and correspondingly write, e.g., $(\mathsf{st}_S, \mathbf{c}) \xleftarrow{\$} \mathsf{aSend}(\mathsf{st}_{S,0}, \mathbf{m})$ to indicate that the sending algorithm is invoked sequentially on input the components of $\mathbf{m}$ and that it outputs as ciphertexts the components of $\mathbf{c}$.

Intuitively, correctness requires that an atomic-message channel recovers the sequence of messages sent as long as the entire sequence of ciphertexts sent is received completely, independently of its fragmentation. It also requires that, if any *prefix* of the sequence of sent ciphertexts is processed at the receiver, then the corresponding prefix of the sent message sequence is recovered completely.

Definition 11.2 (Correctness of atomic-message channels)**.** *Let* $\mathsf{aCh} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be an atomic-message channel. We say that* aCh *provides* correctness *if for all choices of the randomness for algorithms* aInit *and* aSend*, all $\ell, \ell' \geq 0$, all message vectors $\mathbf{m} \in \mathcal{M}^\ell$, all sending output sequences $(\mathsf{st}_{S,\ell}, \mathbf{c}) \xleftarrow{\$} \mathsf{aSend}(\mathsf{st}_{S,0}, \mathbf{m})$, all ciphertext-fragment vectors $\mathbf{c}' \in (\{0,1\}^*)^{\ell'}$, all receiving output sequences $(\mathsf{st}'_{R,\ell'}, \mathbf{m}') \leftarrow \mathsf{aRecv}(\mathsf{st}_{R,0}, \mathbf{c}')$, and every $0 \leq i \leq \ell$, we have*

$$\|\mathbf{c}[1,\ldots,i] \preccurlyeq \|\mathbf{c}' \preccurlyeq \|\mathbf{c} \Longrightarrow \mathbf{m}[1,\ldots,i] \preccurlyeq \mathbf{m}' \preccurlyeq \mathbf{m}.$$

Note that, in contrast to [BDPS12, ADHP16], for correctness in the above setting we do not demand that already $\|\mathbf{c}[1,\ldots,i] \preccurlyeq \|\mathbf{c}'$ implies $\mathbf{m}[1,\ldots,i] \preccurlyeq \mathbf{m}'$, even if $\|\mathbf{c}' \not\preccurlyeq \|\mathbf{c}$. As we already discussed in the streaming setting (cf. Remark 10.4), this would encode a certain amount of robustness in an adversarial setting which is concerned with security rather than correctness.

$\underline{\mathsf{Expt}^{\mathsf{aIND\text{-}ATK},b}_{\mathsf{aCh},\mathcal{A}}(1^\lambda)}$:
1 $(\mathsf{st}_S, \mathsf{st}_R) \xleftarrow{\$} \mathsf{aInit}(1^\lambda)$
2 $\mathsf{sync} \leftarrow 1$
3 $i \leftarrow 0,\ j \leftarrow 1$
4 $\mathbf{M_S} \leftarrow (),\ \mathbf{C_S} \leftarrow ()$
5 $\mathbf{M_R} \leftarrow (),\ C_R \leftarrow \varepsilon$
6 $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}},[\mathcal{O}_{\mathsf{Recv}}]_{\mathsf{ATK=CCFA}}}(1^\lambda)$
7 return b'

$\underline{\mathcal{O}_{\mathsf{LoR}}(m_0, m_1)}$:
8 if $|m_0| \neq |m_1|$ then
9 return $\lightning$
10 $(\mathsf{st}_S, c) \xleftarrow{\$} \mathsf{aSend}(\mathsf{st}_S, m_b)$
11 $i \leftarrow i + 1$
12 $\mathbf{M_S}[i] \leftarrow m_b,\ \mathbf{C_S}[i] \leftarrow c$
13 return c

$\underline{\mathcal{O}_{\mathsf{Recv}}(c)}$:
14 $(\mathsf{st}_R, \mathbf{m}) \leftarrow \mathsf{aRecv}(\mathsf{st}_R, c)$
15 $C_R \leftarrow C_R \| c$
16 $\mathbf{M_R} \leftarrow \mathbf{M_R} \| \mathbf{m}$
17 if $\mathsf{sync} = 0$ then // already out-of-sync
18 return $\mathbf{m}$
19 else if $C_R \preccurlyeq \|\mathbf{C_S}$ then // still in-sync
20 return ()
21 else
22 while $j \leq i$ and $\|\mathbf{C_S}[1, \ldots, j] \preccurlyeq C_R$ and $\mathbf{M_S}[1, \ldots, j] \preccurlyeq \mathbf{M_R}$
23 do $j \leftarrow j + 1$
24 if $j \leq i$ or $|\mathbf{M_R}| > i$ then
// deviation, or exceeding portion produces output
25 $\mathsf{sync} \leftarrow 0$
26 if $j > |\mathbf{M_R}|$ then
27 $\mathbf{m}' \leftarrow ()$
28 else
29 $\mathbf{m}' \leftarrow \mathbf{M_R}[j, \ldots, |\mathbf{M_R}|]$
30 return $\mathbf{m}'$

Figure 11.1: Security experiment for *confidentiality* ($\mathsf{aIND\text{-}ATK}$ with $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCFA}\}$) of atomic-message channels. The brackets $[\mathcal{O}_{\mathsf{Recv}}]_{\mathsf{ATK=CCFA}}$ indicate that only the $\mathsf{aIND\text{-}CCFA}$ adversary has access to the $\mathcal{O}_{\mathsf{Recv}}$ oracle.

11.3 Security of Atomic-Message Channels Supporting Fragmentation

In this section we formalize confidentiality and integrity notions for atomic-message channels supporting fragmentation.

11.3.1 Confidentiality

In order to translate the standard confidentiality requirements against chosen-plaintext attacks and chosen-ciphertext attacks to the setting of atomic-message channels supporting fragmentation we formulate the notions of atomic-message indistinguishability under chosen-plaintext attacks ($\mathsf{aIND\text{-}CPA}$) as well as under chosen ciphertext-fragment attacks ($\mathsf{aIND\text{-}CCFA}$). The former provides the adversary with a left-or-right sending oracle defined in the natural way. The latter essentially transcribes the $\mathsf{IND\text{-}sfCFA}$ notion by Boldyreva et al. [BDPS12] to our setting for atomic-message channels. Briefly, the decryption mechanism of our $\mathsf{aIND\text{-}CCFA}$ notion returns to the adversary all non-genuine message blocks output on receiving the first deviating ciphertext fragment and all follow-up calls.

In more detail, the adversary is provided with a left-or-right oracle $\mathcal{O}_{\mathsf{LoR}}$ and, in the $\mathsf{aIND\text{-}CCFA}$ experiment, with a receiving oracle $\mathcal{O}_{\mathsf{Recv}}$. Left-or-right queries do not include a flush flag (as for stream-based channels); this is a consequence of the syntax. A major difference with the streaming setting is that, since the receiving algorithm outputs atomic-message sequences rather than portions of a stream, synchronization is lost at the ciphertext boundaries (similarly to the case of symmetric encryption supporting fragmentation [BDPS12]). That is, the exact point where synchronization is lost is not necessarily aligned with its counterpart in the streaming setting. However, in contrast to symmetric encryption supporting fragmentation, here suppressing from the received message sequence the (vector) prefix covered by correctness would be

inaccurate, as we explain next.

Suppose that $\mathcal{A}$ causes $\mathcal{O}_{\mathsf{LoR}}$ to send messages $m_1^b, \ldots, m_i^b$ and obtains challenge ciphertexts $c_1, \ldots, c_i$. For the sake of exposition let us make the first out-of-sync ciphertext coincide with the first receiving query and suppose that $\mathcal{A}$ submits to $\mathcal{O}_{\mathsf{Recv}}$ a single ciphertext fragment c^* that contains $c_1 \| \cdots \| c_{j-1}$ as a prefix, for some $j \leq i$, but deviates from c_j onwards. Correctness then does not impose any requirement on the decryption $\mathbf{m}^*$ of the *adversarially* chosen c^*: it may contain none of the messages sent, but it may also contain the full sequence $(m_1^b, \ldots, m_{j-1}^b)$. Thus, despite c^* being out-of-sync, giving $\mathcal{A}$ the full decryption of c^* could lead to trivial wins. However, suppressing from $\mathbf{m}^*$ its first $j-1$ components may hide non-genuine messages, excluding valid attacks from being caught. Intuitively, we want to suppress from the sequence of received messages only the longest genuine prefix. This intuition explains the working principle of the oracle $\mathcal{O}_{\mathsf{Recv}}$, which answers the first out-of-sync query by returning the sequence obtained from $\mathbf{M_R}$ by stripping off the longest genuine prefix $\mathbf{M_S}[1, \ldots, j-1]$ (see lines 21ff. in Figure 11.1), and subsequent out-of-sync queries by returning the full output of aRecv. As for the streaming setting, we define synchronization to be lost only on an exceeding ciphertext fragment if that fragment produces (non-empty) output.

Definition 11.3 ($\mathsf{aIND\text{-}CPA}$ and $\mathsf{aIND\text{-}CCFA}$ Security)**.** *Let* $\mathsf{aCh} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be an atomic-message channel supporting fragmentation and* $\mathsf{Expt}^{\mathsf{aIND\text{-}ATK},b}_{\mathsf{aCh},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *and a bit* b *be defined as in Figure 11.1, where* ATK *is a placeholder for either* CPA *or* CCFA*.*

We say that aCh *provides* atomic-message indistinguishability under chosen-plaintext attacks*, respectively,* chosen ciphertext-fragment attacks *(*$\mathsf{aIND\text{-}CPA}$ *resp.* $\mathsf{aIND\text{-}CCFA}$*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible:*

$$\mathsf{Adv}^{\mathsf{aIND\text{-}ATK}}_{\mathsf{aCh},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{aIND\text{-}ATK},1}_{\mathsf{aCh},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{aIND\text{-}ATK},0}_{\mathsf{aCh},\mathcal{A}}(1^\lambda) = 1\right].$$

11.3.2 Integrity

As for confidentiality we adapt the integrity notions from the streaming setting and define (atomic-message) integrity of plaintexts ($\mathsf{aINT\text{-}PTXT}$) and integrity of ciphertext streams ($\mathsf{aINT\text{-}CST}$). The idea behind plaintext integrity is that the adversary wins if it manages to make the receiver output a valid message sequence that differs from the sequence of messages that have been sent. In integrity of ciphertext streams, instead, the adversary wins if it submits a modified ciphertext sequence for decryption whose deviating part produces some valid messages. Again, our $\mathsf{aINT\text{-}CST}$ notion is analogous to the corresponding $\mathsf{INT\text{-}sfCTF}$ notion by Albrecht et al. [ADHP16] for symmetric encryption supporting fragmentation, proposed concurrently to our work.

In both integrity experiments the adversary is provided with a sending oracle $\mathcal{O}_{\mathsf{Send}}$ and a receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ that it can query on arbitrary messages, respectively, ciphertext fragments. The $\mathsf{aINT\text{-}PTXT}$ experiments declares $\mathcal{A}$ successful if the sequence $\mathbf{M_R}$ of received messages deviates from the sequence $\mathbf{M_S}$ of sent messages and if the deviation contains more than just errors. In the $\mathsf{aINT\text{-}CST}$ experiment the adversary wins if the string C_R submitted (in an arbitrary fragmented way) for decryption deviates from the concatenation $\|\mathbf{C_S}$ of ciphertexts that have been sent and if such deviation causes aRecv to output valid (genuine or non-genuine) messages. In different words, ciphertext-stream integrity is violated if aRecv produces any valid message once it lost synchronization. It is worth mentioning that, as for confidentiality, the exact point where synchronization is lost is defined to align with one of the (sent) ciphertext boundaries and, in contrast to the streaming setting, does not coincide with the first deviating bit of ciphertext. To detect where this point falls the $\mathcal{O}_{\mathsf{Recv}}$ oracle uses the same mechanism as in the $\mathsf{aIND\text{-}CCFA}$ experiment (see lines 25ff. of Figure 11.2).

$\underline{\mathsf{Expt}^{\mathsf{aINT\text{-}ATK}}_{\mathsf{aCh},\mathcal{A}}(1^\lambda)}$:
1 $(\mathsf{st}_S, \mathsf{st}_R) \xleftarrow{\$} \mathsf{aInit}(1^\lambda)$
2 $\mathsf{sync} \leftarrow 1$, $\mathsf{win} \leftarrow 0$
3 $i \leftarrow 0$, $j \leftarrow 1$
4 $\mathbf{M_S} \leftarrow ()$, $\mathbf{C_S} \leftarrow ()$
5 $\mathbf{M_R} \leftarrow ()$, $C_R \leftarrow \varepsilon$
6 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$
7 return win

$\underline{\mathcal{O}_{\mathsf{Send}}(m)}$:
8 $(\mathsf{st}_S, c) \xleftarrow{\$} \mathsf{aSend}(\mathsf{st}_S, m)$
9 $i \leftarrow i + 1$
10 $\mathbf{M_S}[i] \leftarrow m$
11 $\mathbf{C_S}[i] \leftarrow c$
12 return c

$\underline{\mathsf{aINT\text{-}PTXT}\ \mathcal{O}_{\mathsf{Recv}}(c)}$:
13 $(\mathsf{st}_R, \mathbf{m}) \leftarrow \mathsf{aRecv}(\mathsf{st}_R, c)$
14 $\mathbf{M_R} \leftarrow \mathbf{M_R} \| \mathbf{m}$
15 if $\mathbf{M_R} \,\%\, [\mathbf{M_R}, \mathbf{M_S}] \notin \mathcal{E}^*$ then
16 $\quad \mathsf{win} \leftarrow 1$
17 return $\mathbf{m}$

$\underline{\mathsf{aINT\text{-}CST}\ \mathcal{O}_{\mathsf{Recv}}(c)}$:
18 $(\mathsf{st}_R, \mathbf{m}) \leftarrow \mathsf{aRecv}(\mathsf{st}_R, c)$
19 $C_R \leftarrow C_R \| c$
20 $\mathbf{M_R} \leftarrow \mathbf{M_R} \| \mathbf{m}$
21 if $\mathsf{sync} = 0$ then // already out-of-sync
22 $\quad$ if $\mathbf{m} \notin \mathcal{E}^*$ then
23 $\quad\quad \mathsf{win} \leftarrow 1$
24 else if $C_R \not\preccurlyeq \|\mathbf{C_S}$ then // deviating or exceeding
25 $\quad$ while $j \leq i$ and $\|\mathbf{C_S}[1, \ldots, j] \preccurlyeq C_R$ and $\mathbf{M_S}[1, \ldots, j] \preccurlyeq \mathbf{M_R}$
26 $\quad\quad$ do $j \leftarrow j + 1$
27 $\quad$ if $j \leq i$ or $|\mathbf{M_R}| > i$ then
$\quad\quad$ // deviation, or exceeding portion produces output
28 $\quad\quad \mathsf{sync} \leftarrow 0$
29 $\quad$ if $j > |\mathbf{M_R}|$ then
30 $\quad\quad \mathbf{m}' \leftarrow ()$
31 $\quad$ else
32 $\quad\quad \mathbf{m}' \leftarrow \mathbf{M_R}[j, \ldots, |\mathbf{M_R}|]$
33 $\quad$ if $\mathbf{m}' \notin \mathcal{E}^*$ then
34 $\quad\quad \mathsf{win} \leftarrow 1$
35 return $\mathbf{m}$

Figure 11.2: Security experiment for *integrity* ($\mathsf{aINT\text{-}ATK}$ with $\mathsf{ATK} \in \{\mathsf{PTXT}, \mathsf{CST}\}$) of atomic-message channels. A PTXT-attacker is provided with access to the left $\mathcal{O}_{\mathsf{Recv}}$ oracle (INT-PTXT), whereas a CST-attacker is instead granted access to the oracle on the right-hand side (INT-CST).

Definition 11.4 ($\mathsf{aINT\text{-}PTXT}$ and $\mathsf{aINT\text{-}CST}$ Security)**.** *Let* $\mathsf{aCh} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be an atomic-message channel supporting fragmentation and* $\mathsf{Expt}^{\mathsf{aINT\text{-}ATK}}_{\mathsf{aCh},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 11.2, where* ATK *is a placeholder for either* PTXT *or* CST*.*

We say aCh *provides* atomic-message integrity of plaintexts*, respectively,* ciphertext streams *(*$\mathsf{aINT\text{-}PTXT}$ *resp.* $\mathsf{aINT\text{-}CST}$*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible:*

$$\mathsf{Adv}^{\mathsf{aINT\text{-}ATK}}_{\mathsf{aCh},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{aINT\text{-}ATK}}_{\mathsf{aCh},\mathcal{A}}(1^\lambda) = 1\right].$$

11.3.3 Relations Amongst Notions and Composition

Here we explore how the different security notions for atomic-message channels supporting fragmentation relate to each other. Similarly to the case of stream-based channels (cf. Section 10.3.3), we can show that integrity of ciphertext streams implies integrity of plaintext streams, that indistinguishability against chosen-ciphertext attacks implies indistinguishability against chosen-plaintext attacks, and that the weaker confidentiality notion together with ciphertext integrity and (an adaptation of) error predictability imply the stronger confidentiality notion.

Confidentiality. The relation $\mathsf{aIND\text{-}CCFA} \Longrightarrow \mathsf{aIND\text{-}CPA}$ immediately follows from the fact that the $\mathsf{aIND\text{-}CCFA}$ experiment gives access to a left-or-right oracle as in the $\mathsf{aIND\text{-}CPA}$ experiment as well as to a decryption oracle. In particular, a successful chosen-plaintext attack can be seen as a successful chosen ciphertext-fragment attack that ignores the decryption oracle.

Integrity. To see that aINT-CST $\Longrightarrow$ aINT-PTXT it suffices to observe that, in order to produce a deviation in the sequence of messages accepted by aRecv an adversary must submit for decryption a sequence of ciphertext fragments that, in turn, deviates from the genuine ciphertext sequence so that the deviating part produces some valid messages. Put differently, an adversary cannot violate the plaintext integrity property without violating integrity of ciphertext streams. We formalize this intuition in the following proposition.

Proposition 11.5 (aINT-CST $\Longrightarrow$ aINT-PTXT)**.** *Let* aCh = (aInit, aSend, aRecv) *be a correct atomic-message channel supporting fragmentation. If* aCh *provides integrity of ciphertext streams then it also provides integrity of plaintexts and, in particular,* $\mathsf{Adv}^{\mathsf{aINT\text{-}PTXT}}_{\mathsf{aCh},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{aINT\text{-}CST}}_{\mathsf{aCh},\mathcal{A}}(\lambda)$ *for any adversary* $\mathcal{A}$*.*

Proof. Consider an execution of the aINT-PTXT experiment with an adversary $\mathcal{A}$ against the channel aCh. Since the aINT-PTXT and aINT-CST experiments both provide interfaces to a sending oracle $\mathcal{O}_{\mathsf{Send}}$ and a receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ we can imagine to run the two experiments simultaneously and show that if $\mathcal{A}$ is successful in the former, so is in the latter. More specifically, we show that if the aINT-PTXT experiment sets win $\leftarrow 1$ in line 16 then the aINT-CST experiment sets win $\leftarrow 1$ in lines 23 or 34.

Observe that $\mathcal{A}$ triggers the execution of line 16 (setting win to 1) if it submits some ciphertext fragments to $\mathcal{O}_{\mathsf{Recv}}$ in experiment aINT-PTXT that result in a message sequence $\mathbf{M_R}$ deviating from the genuine message sequence $\mathbf{M_S}$, beyond errors. By correctness, a deviation in the message sequence can only originate from a deviation in the ciphertext sequence and, thus, we know that $\mathcal{A}$ submits at least one out-of-sync ciphertext. Suppose that $\mathcal{A}$ triggers the execution of line 16 of the aINT-PTXT experiment already when it submits the first out-of-sync ciphertext: then such ciphertext causes the execution of line 28 and the following loop in the aINT-CST experiment, yielding a vector $\mathbf{m}'$ that contains all messages received so far but the longest common prefix with the sequence of sent messages. It follows from the assumptions made that $\mathbf{m}'$ contains some valid message and, thus, win $\leftarrow 1$ is set in line 34 in the aINT-CST experiment, too. Suppose now that $\mathcal{A}$ enforces the execution of line 16 of the aINT-PTXT experiment after the first out-of-sync ciphertext has been submitted. Then we have two possibilities: either the deviating part of the first out-of-sync ciphertext fragment produces some valid messages and, thus, in the aINT-CST experiment sets win $\leftarrow 1$ in line 34, or some of the following ciphertext fragments that $\mathcal{A}$ submits cause aRecv to output valid messages, hence causing aINT-CST to set win $\leftarrow 1$ in line 23. □

Composition. As in the case of stream-based channels, we can prove that confidentiality against passive adversaries in combination with ciphertext-stream integrity can be lifted to confidentiality against active adversaries by additionally requiring that decryption errors are efficiently predictable. For this we adapt our stream-based error predictability notion (see Definition 10.9 on page 161) to the atomic-message setting in the natural way by demanding the predictor to output the vector of errors that aRecv would return on input a given sequence of ciphertext fragments. More formally, we say that an atomic-message channel provides (atomic-message) *error predictability* (aERR-PRE) with respect to an efficient probabilistic *predictor* algorithm Pred if this predictor Pred, given the vector $\mathbf{C_S}$ of ciphertexts sent, the string C_R of ciphertext fragments received-so-far, and the 'next' ciphertext fragment c, returns a (potentially empty) vector containing all the errors that aRecv would output on input the (arbitrarily fragmented) string $C_R \| c$, in the same order they appear when output by the latter.

Definition 11.6 (Atomic-message error predictability (aERR-PRE))**.** *Let* aCh = (aInit, aSend, aRecv) *be an atomic-message channel supporting fragmentation with message space* $\mathcal{M}$ *and error*

$\underline{\mathsf{Expt}^{\mathsf{aERR\text{-}PRE}}_{\mathsf{aCh},\mathsf{Pred},\mathcal{A}}(1^\lambda)}$:
1 $(\mathsf{st}_S, \mathsf{st}_R) \xleftarrow{\$} \mathsf{aInit}(1^\lambda)$
2 $\mathsf{win} \leftarrow 0$, $i \leftarrow 0$
3 $\mathbf{C_S} \leftarrow ()$
4 $C_R \leftarrow \varepsilon$
5 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$
6 return win

$\underline{\mathcal{O}_{\mathsf{Send}}(m)}$:
7 $(\mathsf{st}_S, c) \xleftarrow{\$} \mathsf{aSend}(\mathsf{st}_S, m)$
8 $i \leftarrow i + 1$
9 $\mathbf{C_S}[i] \leftarrow c$
10 return c

$\underline{\mathcal{O}_{\mathsf{Recv}}(c)}$:
11 $(\mathsf{st}_R, \mathbf{m}) \leftarrow \mathsf{aRecv}(\mathsf{st}_R, c)$
12 if $\langle \mathbf{m} \rangle_{\mathcal{E}} \neq \mathsf{Pred}(\mathbf{C_S}, C_R, c)$ then
13 $\quad \mathsf{win} \leftarrow 1$
14 $C_R \leftarrow C_R \| c$
15 return $\mathbf{m}$

Figure 11.3: Security experiment for *error predictability* (aERR-PRE) of atomic-message channels. We denote by $\langle \cdot \rangle_{\mathcal{E}} \colon (\mathcal{M} \cup \mathcal{E})^* \to \mathcal{E}^*$ the 'projection on the error space', i.e., the mapping that removes from a vector all occurrences that do not belong to the error space $\mathcal{E}$. For instance, if $\mathbf{m} = (m_1, \perp_1, m_2, m_3, \perp_2)$ with $m_1, m_2, m_3 \in \mathcal{M}$ then $\langle \mathbf{m} \rangle_{\mathcal{E}} = (\perp_1, \perp_2)$.

space $\mathcal{E}$, and let Pred *be an efficient probabilistic algorithm. We say that* aCh *provides* (atomic-message) error predictability (aERR-PRE) with respect to Pred *if for every PPT adversary $\mathcal{A}$ playing in the experiment* aERR-PRE *defined in Figure 11.3 against channel* aCh*, the following advantage function is negligible:*

$$\mathsf{Adv}^{\mathsf{aERR\text{-}PRE}}_{\mathsf{aCh},\mathsf{Pred},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{aERR\text{-}PRE}}_{\mathsf{aCh},\mathsf{Pred},\mathcal{A}}(1^\lambda) = 1\right].$$

We are now ready to state a composition result for atomic-message channels analogous to that for stream-based channels.

Theorem 11.7 (aINT-CST $\wedge$ aIND-CPA $\wedge$ aERR-PRE $\Longrightarrow$ aIND-CCFA)**.** *Let* aCh *be a (correct) atomic-message channel supporting fragmentation. If* aCh *provides integrity of ciphertext streams (*aINT-CST*), indistinguishability against chosen-plaintext attacks (*aIND-CPA*), as well as error predictability (*aERR-PRE*) with respect to a predictor* Pred*, then it also provides indistinguishability against chosen ciphertext-fragment attacks (*aIND-CCFA*).*

To prove this relation we can apply essentially the same strategy used in the proof of Theorem 10.10 (page 161) and, for this reason, we abstain from providing a full proof but recall the informal argument. Assume that we have an adversary $\mathcal{A}$ attacking the aIND-CCFA property of a channel that provides aIND-CPA, aINT-CST, and aERR-PRE. Then given only CPA capabilities one can answer $\mathcal{A}$'s queries by forwarding sending queries to the left-or-right oracle provided by the aIND-CPA experiment, returning empty vectors in response to in-sync decryption queries, and returning the output of the error predictor on input out-of-sync decryption queries. The aINT-CST property ensures that no valid message originates from out-of-sync decryption queries, while the aERR-PRE property allows to (efficiently) compute decryption errors.

11.4 Generic Construction of Atomic-Message Channels from Stream-Based Channels

In practice, applications relying on atomic-message processing perform some encoding of those messages prior to handing them over to the underlying (stream-based) secure channel. The HTTP protocol provides two prime examples for such encoding approaches. HTTP headers are encoded by having an empty line indicate the end of the header section [FR14, Section 3], i.e., a header message ends with the distinguished "end-of-message" marker `"\n\n"` which is not allowed to occur anywhere else in the header.[57] An HTTP body message in contrast can be an

[57] As a technical side remark, violating the HTTP header decoding rules was part of what enabled the 'cookie-cutter' attack [BDF+14].

arbitrary byte string (i.e., the specification cannot single out a distinguished end-of-message symbol) and is hence, as one option, demarcated through indicating the body length in the `Content-Length` header field [FR14, Section 3.3]. Of course there are numerous alternative approaches to encode atomic messages in a bit stream; prepending the message with a fixed-length binary encoding of its length is a particularly efficient one, applicable whenever (an upper bound on) the message length is known.

For our generic construction that enables secure transmission of atomic messages over a generic secure stream-based channel we capture all these and further approaches under the framework of instantaneously decodable encodings.

11.4.1 Length-Regular Instantaneously Decodable Encoding Schemes

We recall the properties of *length-regular instantaneously decodable encoding schemes* that will be later used as a tool in our construction. The idea of using instantaneously decodable encodings was already employed by Boldyreva et al. [BDPS12] in the context of symmetric encryption supporting fragmentation in order to encode (atomic) messages in ciphertexts that might be fragmented. Such an encoding consists of an algorithm Encode that turns a word w into a codeword v, and an algorithm Decode which takes a string as input and outputs a vector $\mathbf{w}$ of words and a string s (the latter is, in fact, the part of the input string which contains no full words as a prefix).

Definition 11.8 (Length-regular instantaneously decodable encoding schemes)**.** *An* encoding scheme *with word space $W \subseteq \{0,1\}^*$ is a pair $\mathsf{ES} = (\mathsf{Encode}, \mathsf{Decode})$ of efficient deterministic algorithms defined as follows. The encoding algorithm* Encode *takes as input a word $w \in W$ and returns a codeword $v \leftarrow \mathsf{Encode}(w)$ where $v \in \{0,1\}^*$. The decoding algorithm* Decode *takes as input a string $v' \in \{0,1\}^*$ and outputs a (potentially empty) vector of words $\mathbf{w}' \in W^*$ and a string $s' \in \{0,1\}^*$. We indicate this by writing $(\mathbf{w}', s') \leftarrow \mathsf{Decode}(v')$. We use the shorthand $\mathbf{v} \leftarrow \mathsf{Encode}(\mathbf{w})$ to indicate that* Encode *is executed sequentially on the components of $\mathbf{w} = (w_1, \ldots, w_n) \in W^*$ and the corresponding codewords are the components of $\mathbf{v} = (v_1, \ldots, v_n) \in (\{0,1\}^*)^*$ with $v_i \leftarrow \mathsf{Encode}(w_i)$.*

We say that ES *is* instantaneously decodable *if for all $\mathbf{w} \in W^*$ and $s \in \{0,1\}^*$, and for $\mathbf{v} \leftarrow \mathsf{Encode}(\mathbf{w})$ and $(\mathbf{w}', s') \leftarrow \mathsf{Decode}(v_1 \| \ldots \| v_n \| s)$ where $\mathbf{v} = (v_1, \ldots, v_n)$, the following two properties hold:*

ID1. $\mathbf{w} \preccurlyeq \mathbf{w}'$*, i.e., all input words from* $\mathbf{w}$ *are recovered by* Decode *in* $\mathbf{w}'$ *(and potentially further words contained in the string s), and*

ID2. If there is no $w \in W$ such that $\mathsf{Encode}(w) \preccurlyeq s$ then $\mathbf{w}' = \mathbf{w}$ and $s' = s$, i.e., if s does not contain an encoded word then Decode *recovers exactly the words in $\mathbf{w} = \mathbf{w}'$ and puts the remaining bits in s'.*

We furthermore say that ES *is* length-regular *if for all $\mathbf{w}'$, $\mathbf{w}$ with $|\mathbf{w}'| = |\mathbf{w}|$ it holds that $|\mathbf{v}| = |\mathbf{v}'|$ where $\mathbf{v} \leftarrow \mathsf{Encode}(\mathbf{w})$ and $\mathbf{v}' \leftarrow \mathsf{Encode}(\mathbf{w}')$.*

Since in this chapter we only make use of encoding schemes that are instantaneously decodable and length-regular, from now on the term 'encoding scheme' refers to schemes fulfilling these properties.

Remark 11.9. For every $\mathbf{w} \in W^*$ and $\mathbf{v} \leftarrow \mathsf{Encode}(\mathbf{w})$ it holds that $\mathsf{Decode}(\|\mathbf{v}) = (\mathbf{w}, \varepsilon)$. Indeed, for $\mathbf{v} = (v_1, \ldots, v_n)$, $s = \varepsilon$ and $(\mathbf{w}', s') \leftarrow \mathsf{Decode}(v_1 \| \cdots \| v_n \| s)$ property (ID2) implies $\mathbf{w}' = \mathbf{w}$ and $s' = \varepsilon$.

Remark 11.10. The set of codewords $V = \{\mathsf{Encode}(w) \mid w \in W\}$ induced by ES is *prefix-free*. Indeed, let $v, v' \in V$ be such that $v \preccurlyeq v'$ and write $v' = v\|s$ for some $s \in \{0,1\}^*$. By definition there exist $w, w' \in W$ such that $v = \mathsf{Encode}(w)$ and $v' = \mathsf{Encode}(w')$. By Remark 11.9 we have $\mathsf{Decode}(v') = ((w'), \varepsilon)$; similarly, by property (ID2) we derive $\mathsf{Decode}(v') = \mathsf{Decode}(v\|s) = (\mathbf{w}'', s'')$ where $(w) \preccurlyeq \mathbf{w}''$. Putting these relations together we get $(w) \preccurlyeq \mathbf{w}'' = (w') \implies w = w'$ and thus $v = \mathsf{Encode}(w) = \mathsf{Encode}(w') = v'$. In what follows, when writing that an encoding scheme is prefix-free we mean that its set of codewords is.

Example 11.11 (The end-of-message encoding). Take any string $\diamond \in \{0,1\}^*$ and let $\kappa = |\diamond|$ be its length. Define $W \subset \{0,1\}^*$ recursively in such a way that no (finite) concatenation of words in W contains the distinguished string $\diamond$. Formally, we require that for all $u \in W$ with $|u| \geq \kappa$ and for all i such that $1 \leq i \leq |u| - \kappa$ it holds $\diamond \neq u[i, \ldots, i + \kappa]$. We define the *end-of-message encoding* through the following functions.

We encode $m \in W$ by appending to it the end-of-message symbol $\diamond$, i.e., $\mathsf{Encode}(m) = m\|\diamond$. To decode a string $y \in \{0,1\}^*$ we first initialize $\mathbf{w} \leftarrow ()$, $s \leftarrow \varepsilon$, then we scan y from left to right until we find the first occurrence of $\diamond$. If there is none, we set $s = y$ and return $(\mathbf{w}, s)$. Otherwise we found the first word w_1 that y encodes, i.e., such that $w_1\|\diamond \preccurlyeq y$; thus, we append w_1 to $\mathbf{w}$ and proceed (recursively) as above using the unprocessed string $y' \leftarrow y \,\%\, (w_1\|\diamond)$ instead of y. Observe that, by definition of W, any string $y \in \{0,1\}^*$ admits a unique decomposition $y = w_1\|\diamond\|\cdots\|w_\ell\|\diamond\|w_{\ell+1}$ with $\ell \geq 0$, $w_1, \ldots, w_\ell \in W$ and $w_{\ell+1} \in \{0,1\}^* \setminus W$. By the obvious correctness of this algorithm, wee see that the end-of-message encoding is instantaneously decodable. It is also length-regular due to the fixed length κ of the appended end-of-message symbol $\diamond$.

A specific instance of the end-of-message encoding is the HTTP header encoding where $\diamond =$ `\n\n` (i.e., two ASCII newline characters) and header messages follow a specific format which in particular forbids two subsequent newlines to occur in a message.

11.4.2 The Encode-then-Stream Construction

For our generic construction of an atomic-message channel from a stream-based channel we now leverage a length-regular instantaneously decodable encoding scheme $\mathsf{ES} = (\mathsf{Encode}, \mathsf{Decode})$ with word space $W = \mathcal{M}$ as a generalization of both real-world and theoretical approaches to convert atomic messages into a stream and, vice versa, a stream into (a vector of) atomic messages. We name this paradigm *encode-then-stream* and denote by $\mathsf{aCh}_{\mathsf{EtS}}$ the resulting atomic-message channel.

The main idea is very natural: we encode atomic messages within the message stream sent and, on the receiver's side, to buffer the incoming stream and only output messages once they are received completely. Briefly, algorithm aSend takes an atomic message m and first encodes it by invoking $v \leftarrow \mathsf{Encode}(m)$. It then processes the corresponding string v using the stream-based channel's sending algorithm (with a flush request), obtaining $(\mathsf{st}_S, c) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}_S, v, 1)$. Correspondingly, algorithm aRecv takes as input a ciphertext fragment $c \in \{0,1\}^*$ and first invokes the streaming receiving algorithm $(\mathsf{st}_R, v) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$. It then extracts from v the longest prefix v' that does not contain error symbols and concatenates the latter to the buffer, $\mathsf{buf} \leftarrow \mathsf{buf}\|v'$. Finally, it decodes the new buffer content to obtain an atomic-message vector and an updated (potentially empty) buffer, $(\mathbf{m}, \mathsf{buf}) \leftarrow \mathsf{Decode}(\mathsf{buf})$.

Construction 11.12 (Encode-then-stream construction $\mathsf{aCh}_{\mathsf{EtS}}$)**.** *Consider a stream-based channel* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *with error space* $\mathcal{E}$ *and an encoding scheme* $\mathsf{ES} = (\mathsf{Encode}, \mathsf{Decode})$ *with word space* $W \subseteq \{0,1\}^*$*. We define* $\mathsf{aCh}_{\mathsf{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *to be the atomic-message channel with message space* $\mathcal{M} = W$ *and error space* $\{\bot\}$ *obtained by applying to* Ch *the transform described in Figure 11.4.*

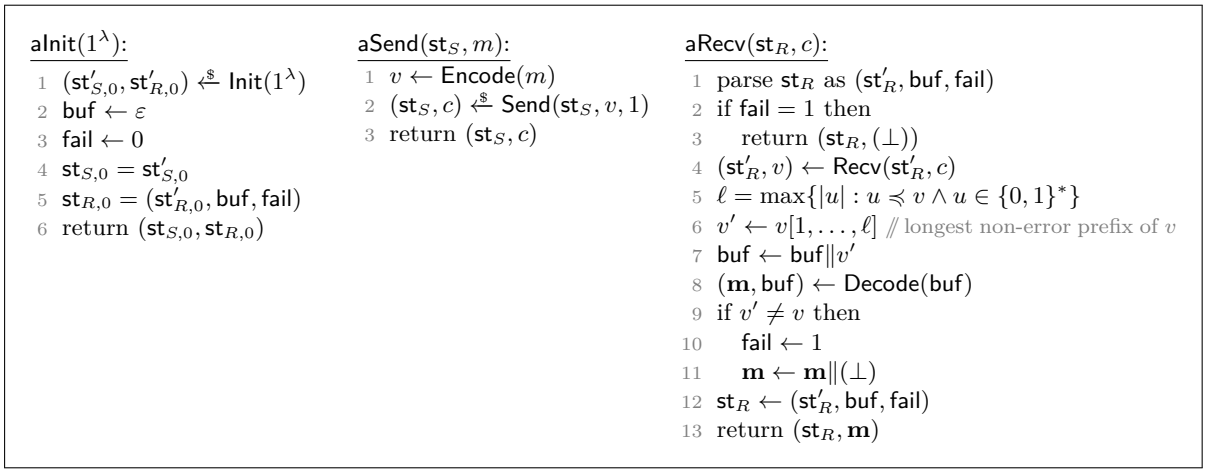


Figure 11.4: Generic construction of an atomic-message channel $\mathsf{aCh}_{\mathsf{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ with message space $\mathcal{M}$ from any stream-based channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ and an encoding scheme $\mathsf{ES} = (\mathsf{Encode}, \mathsf{Decode})$ with word space $W = \mathcal{M}$.

Correctness of $\mathsf{aCh}_{\mathsf{EtS}}$ directly follows from the correctness of Ch and the instantaneous decodability of ES, as we show in the following proposition.

Proposition 11.13 (Correctness of $\mathsf{aCh}_{\mathsf{EtS}}$)**.** *If the stream-based channel* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *is correct and the encoding scheme* $\mathsf{ES} = (\mathsf{Encode}, \mathsf{Decode})$ *is instantaneously decodable then the atomic-message channel* $\mathsf{aCh}_{\mathsf{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *is correct, too.*

Proof. Let $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0}) \xleftarrow{\$} \mathsf{aInit}(1^\lambda)$, $\ell \in \mathbb{N}$, and $\mathbf{m} \in \mathcal{M}^\ell$ be arbitrary and let $\mathbf{c} \in (\{0,1\}^*)^\ell$ be such that $(\mathsf{st}_{S,\ell}, \mathbf{c}) \xleftarrow{\$} \mathsf{aSend}(\mathsf{st}_{S,0}, \mathbf{m})$. Let $\ell' \in \mathbb{N}$ and $\mathbf{c}' \in (\{0,1\}^*)^{\ell'}$ be arbitrary, and let $\mathbf{m}' \in \mathcal{M}^{\ell'}$ be such that $(\mathsf{st}_{R,\ell'}, \mathbf{m}') \leftarrow \mathsf{aRecv}(\mathsf{st}_{R,0}, \mathbf{c}')$. Suppose that for some $i \in [1, \ldots, \ell]$ it holds $\|\mathbf{c}[1 \ldots i] \preccurlyeq \|\mathbf{c}' \preccurlyeq \mathbf{c}$. Using a similar notation let $\mathbf{v} \in (\{0,1\}^*)^\ell$ denote the vector of codewords $\mathbf{v} \leftarrow \mathsf{Encode}(\mathbf{m})$ and let $\mathbf{v}' \in (\{0,1\}^*)^{\ell'}$ be such that $(\mathsf{st}'_{R,\ell'}, \mathbf{v}') \leftarrow \mathsf{Recv}(\mathsf{st}'_{R,0}, \mathbf{c}')$. Observe that $\mathbf{c}$ is generated by invoking the stream-based sending algorithm Send on input the components of $\mathbf{v}$ and flush flags $\mathbf{f} = (1, \ldots, 1)$. Then, by (stream-based) correctness of Ch we have that for all $j \in [1, \ldots, \ell]$, and in particular for $j = i$, it holds $\|\mathbf{c}[1, \ldots, j] \preccurlyeq \|\mathbf{c}' \preccurlyeq \|\mathbf{c} \Longrightarrow \|\mathbf{v}[1, \ldots, j] \preccurlyeq \|\mathbf{v}' \preccurlyeq \|\mathbf{v}$. By construction we now have $(\mathbf{m}[1, \ldots, i], \varepsilon) \leftarrow \mathsf{Decode}(\|\mathbf{v}[1, \ldots, i])$, $(\mathbf{m}', s) \leftarrow \mathsf{Decode}(\|\mathbf{v}')$ for some $s \in \{0,1\}^*$ and $(\mathbf{m}, \varepsilon) \leftarrow \mathsf{Decode}(\|\mathbf{v})$. Using the property ID1 of encoding schemes we derive from $\|\mathbf{v}[1, \ldots, i] \preccurlyeq \|\mathbf{v}'$ that $\mathbf{m}[1, \ldots, i] \preccurlyeq \mathbf{m}'$. Furthermore, $\mathbf{m}' \preccurlyeq \mathbf{m}$ as property ID2 ensures that Decode only decodes full codewords in $\|\mathbf{v}' \,\%\, \|\mathbf{v}[1, \ldots, i]$ namely, by property ID1 and as $\|\mathbf{v}' \preccurlyeq \|\mathbf{v}$, those codewords $\mathbf{v}[i+1, \ldots, \ell]$ corresponding to $\mathbf{m}[i+1, \ldots, \ell]$ fully contained in $\|\mathbf{v}'$. □

11.5 Security of the Encode-then-Stream Construction

Ideally we would like to show that confidentiality and integrity properties of the stream-based channel Ch can be lifted to the corresponding security properties of the atomic-message channel $\mathsf{aCh}_{\mathsf{EtS}}$. Indeed, using a specific encoding to identify message boundaries—as our generic construction $\mathsf{aCh}_{\mathsf{EtS}}$ does—is a natural approach to encode atomic messages within a stream of bits. This approach is pursued by numerous application layer protocols including, among others, HTTP [FR14]. Surprisingly, not even the strongest confidentiality and integrity properties of the underlying stream-based channel Ch (i.e., IND-CCFA and INT-CST) suffice to make our atomic-message channel construction $\mathsf{aCh}_{\mathsf{EtS}}$ aIND-CCFA- and aINT-CST-secure.

As a particular (and admittedly artificial) counterexample consider the following variant $\mathsf{Ch}'_{\mathsf{AEAD}} = (\mathsf{Init}', \mathsf{Send}', \mathsf{Recv}')$ of the AEAD-based streaming channel construction $\mathsf{Ch}_{\mathsf{AEAD}}$ (see Construction 10.12): The Send' algorithm processes the input message as Send does, but additionally appends a 0-bit to each AEAD ciphertext c' computed by Send. On the receiver's side, the AEAD ciphertext is processed as before and, if the appended bit following the AEAD ciphertext (which is also allowed to arrive in a separate fragment) equals 1, then after decrypting the ciphertext and adding the result to the output message m the failure flag is set to $\mathsf{fail} \leftarrow 1$ and the error symbol $\perp$ is appended to m.

Observe that construction $\mathsf{Ch}'_{\mathsf{AEAD}}$ preserves the IND-CCFA and INT-CST security properties of the original stream-based channel $\mathsf{Ch}_{\mathsf{AEAD}}$ (cf. Section 10.4) since, intuitively, the ability to flip the redundant bit allows the adversary only to create an extra error symbol which harms neither confidentiality nor integrity. The reason is that the message part from the unaltered ciphertext prefix, without the extra bit, is still suppressed in the stream-based confidentiality experiment. Analogously, creating additional error symbols in the stream-based integrity experiment does not violate security. However, as we show next, using $\mathsf{Ch}'_{\mathsf{AEAD}}$ as the underlying stream-based channel results in the atomic-message construction $\mathsf{aCh}_{\mathsf{EtS}}$ being insecure with respect to both confidentiality and integrity, as we discuss next.

Let us consider confidentiality first: for any ciphertext $c = len\|c'\|0$ (where $len\|c'$ is the ciphertext as generated by $\mathsf{Ch}_{\mathsf{AEAD}}$) output by the left-or-right oracle, an adversary against the aIND-CCFA security of $\mathsf{aCh}_{\mathsf{EtS}}$ can simply query the receiving oracle on $c^* = len\|c'\|1$ and will obtain the input message m'_b used on the sender's side. Indeed, as the AEAD ciphertext c' is unmodified, the AEAD decryption will yield the full codeword $v' = \mathsf{Encode}(m'_b)$. Therefore the (atomic-message) receiving oracle $\mathcal{O}_{\mathsf{Recv}}$, treating c^* as an atomic (and hence differing) ciphertext, will return the pair $(m'_b, \perp)$ to the adversary, allowing it to break confidentiality. However, since in the streaming setting the message m'_b is already output upon receiving the genuine part $\widetilde{c^*} = len\|c'$ of the ciphertext c^*, stream-based confidentiality of $\mathsf{Ch}'_{\mathsf{AEAD}}$ is not affected by this attack.

A similar argument applies to the case of integrity: an adversary against the aINT-CST property of $\mathsf{aCh}_{\mathsf{EtS}}$ can, given a ciphertext $c = len\|c'\|0$, flip the last bit of c and make the receiving oracle rate the resulting message m' (where $m' = \mathsf{Decode}(v')$ and v' is the codeword obtained by AEAD decrypting c') as a valid message output on a deviating ciphertext c^*. The latter breaks the atomic ciphertext integrity of $\mathsf{aCh}_{\mathsf{EtS}}$ without violating the notion of stream-based integrity of $\mathsf{Ch}'_{\mathsf{AEAD}}$.

We stress that the modified construction $\mathsf{Ch}'_{\mathsf{AEAD}}$ certainly constitutes a particularly unnatural, yet secure stream-based channel. It nevertheless indicates the need for an additional requirement on the stream-based channel, ruling out such behavior, for proving the security of $\mathsf{aCh}_{\mathsf{EtS}}$ generically.[58] We conjecture that this is not a limitation specific to our $\mathsf{aCh}_{\mathsf{EtS}}$ construction but that indeed no generic atomic-message channel construction working, from a protocol-layering perspective, on top of $\mathsf{Ch}'_{\mathsf{AEAD}}$ in a black-box manner can satisfy confidentiality or integrity. As an attempt to formalize the additional requirement just mentioned, we propose a new security notion that precludes a stream-based channel from behaving like $\mathsf{Ch}'_{\mathsf{AEAD}}$ and prove it sufficient, together with INT-CST and IND-CCFA security, for the security of the

[58] We note that Canetti et al. [CKN03] introduced a relaxed notion of confidentiality (for public-key encryption, but also applicable to the secret-key setting), so-called indistinguishability under replayable chosen-ciphertext attacks (RCCA), capturing that an encryption scheme is non-malleable beyond trivial ciphertext modifications (like our additional, flippable bit). It is plausible that the channel $\mathsf{Ch}_{\mathsf{AEAD}}$ achieves a similar, relaxed notion of confidentiality for stream-based channels (an RCCA-style notion in the streaming setting is yet to be defined, though). Here we take a different route and, aiming at a stronger, CCA-like confidentiality property, we rather make explicit an additional property for stream-based channels—conciseness of ciphertexts—that enables lifting confidentiality and integrity of the underlying stream-based channel to the constructed atomic-message channel.

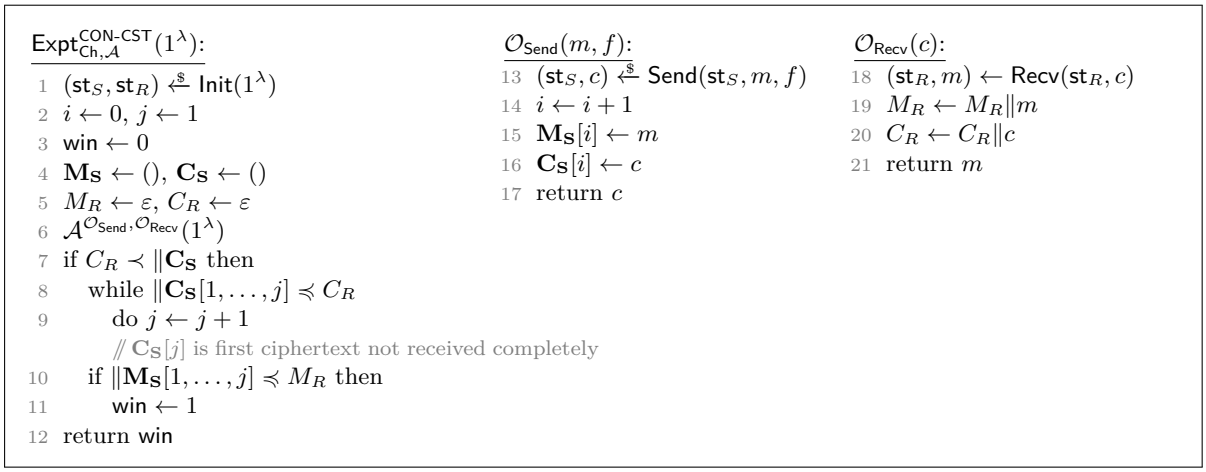


Figure 11.5: Security experiment for *conciseness of ciphertext streams* (CON-CST) for stream-based channels.

encode-then-stream paradigm.

11.5.1 Conciseness of ciphertext streams

Intuitively, we want to ensure that if one submits a *strict prefix* of the genuine ciphertext stream for decryption then the *complete* message stream will not be output on the receiver's side. This, in particular, rules out those constructions for which Send appends redundant bits to the ciphertext fragment (as the construction $\mathsf{Ch}'_{\mathsf{AEAD}}$ in our example above) that can be chopped without affecting the underlying message fragment. Put differently, we require that accepted ciphertexts are concise. We thus name this new security property *conciseness of ciphertext streams* (CON-CST).[59] The intuition behind our definition is as follows. The adversary's goal is to deliver only a strict prefix C_R of the sender's output stream $\|\mathbf{C_S}$ of the atomic ciphertexts to the receiver, but such that the receiver still obtains all message chunks. In other words, the adversary wins if it manages to cut some bits in the ciphertext stream (such as a redundant bit in a ciphertext) without affecting the message delivery. As we will see below, conciseness is naturally achievable by stream-based channel constructions, including ours. Investigating the necessity of conciseness (or a different notion) for sending atomic messages over a stream-based channel in a protocol-layered manner is a possible avenue for future work.

Definition 11.14 (Conciseness of ciphertext streams (CON-CST))**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and experiment* $\mathsf{Expt}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 11.5.*

We say Ch *provides* conciseness of ciphertext streams *(*CON-CST*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible:*

$$\mathsf{Adv}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right].$$

Remark 11.15. It is easy to see that for a stream-based channel with concise ciphertext streams, the Send algorithm can never output a non-empty ciphertext $c \neq \varepsilon$ on input an

[59] The scope of conciseness for stream-based channels is somewhat similar in spirit to that of *tidiness* [NRS14] for nonce-based encryption: it is a natural requirement to rule out schemes for which the receiving algorithm performs some useless operation allowing an adversary to trivially break security.

empty message $m = \varepsilon$ and having no buffered message input from previous calls. Assume otherwise, i.e., a sequence of message fragments $\mathbf{M_S}[1], \ldots, \mathbf{M_S}[i-1], \mathbf{M_S}[i]$ transformed by Send (with the flush flag always set to $f = 1$) into a sequence of ciphertext fragments $\mathbf{C_S}[1], \ldots, \mathbf{C_S}[i-1], \mathbf{C_S}[i]$; with $\mathbf{M_S}[i] = \varepsilon$ while $\mathbf{C_S}[i] \neq \varepsilon$. Now, an adversary can simply query $\mathcal{O}_{\mathsf{Recv}}$ on $\mathbf{C_S}[1] \| \ldots \| \mathbf{C_S}[i-1]$ first, next query $\mathcal{O}_{\mathsf{Recv}}$ on the bit-wise inverse of $\mathbf{C_S}[i]$, and then stop. It wins the $\mathsf{CON\text{-}CST}$ experiment, as $\mathbf{C_S}[i]$ is not contained in the received ciphertext stream C_R, yet all message fragments including $\mathbf{M_S}[i]$ are contained in the received message stream M_R, as $\mathbf{M_S}[1] \| \ldots \| \mathbf{M_S}[i-1] \| \mathbf{M_S}[i] = \mathbf{M_S}[1] \| \ldots \| \mathbf{M_S}[i-1]$ (recall that $\mathbf{M_S}[i] = \varepsilon$) and the latter is contained by correctness.

We remark that in practice channel protocols including IPsec and TLS do specify the possibility to send empty message fragments as a traffic analysis countermeasure; yet, popular libraries (e.g., OpenSSL [Ope]) do not allow this option. The common 'encode-then-stream' approach which we capture in our generic $\mathsf{aCh}_{\mathsf{EtS}}$ construction does not rely on the capability of sending empty message fragments. We hence do consider the restriction to disallow empty message fragments as non-critical in this setting.

11.5.2 Integrity and Confidentiality of Encode-then-Stream

We now turn towards analyzing the security of our generic construction of an atomic-message channel $\mathsf{aCh}_{\mathsf{EtS}}$ from a stream-based channel Ch and an encoding scheme ES. In brief, we prove that integrity of ciphertext streams ($\mathsf{INT\text{-}CST}$) of the stream-based channel Ch can be lifted to the analogous property ($\mathsf{aINT\text{-}CST}$) for the resulting atomic-message channel $\mathsf{aCh}_{\mathsf{EtS}}$, provided that Ch also offers conciseness of ciphertexts ($\mathsf{CON\text{-}CST}$). Due to the subtle difference between the synchronization mechanisms in the streaming and the atomic-message setting, the proof of this result is quite involved. We also show that indistinguishability under chosen plaintext-fragment attacks ($\mathsf{IND\text{-}CPFA}$) and error predictability ($\mathsf{ERR\text{-}PRE}$) of Ch can be lifted to the analogous properties ($\mathsf{aIND\text{-}CPA}$ and $\mathsf{aERR\text{-}PRE}$) of $\mathsf{aCh}_{\mathsf{EtS}}$. These relations together with Theorem 11.7 imply that $\mathsf{INT\text{-}CST}$, $\mathsf{IND\text{-}CPFA}$, $\mathsf{CON\text{-}CST}$ and $\mathsf{ERR\text{-}PRE}$ of Ch are sufficient conditions for the encode-then-stream paradigm to provide indistinguishability under chosen ciphertext-fragment attacks ($\mathsf{IND\text{-}CCFA}$).

It is worth noting that, while integrity of ciphertext streams ($\mathsf{INT\text{-}CST}$), given $\mathsf{CON\text{-}CST}$, directly carries over from the stream-based channel Ch to the atomic-message channel $\mathsf{aCh}_{\mathsf{EtS}}$, the same does not seem to hold for confidentiality against active adversaries ($\mathsf{IND\text{-}CCFA}$). While our proof lifts $\mathsf{aIND\text{-}CPA}$ to $\mathsf{aIND\text{-}CCFA}$ security by leveraging ciphertext-stream integrity and error predictability (via the compositional result from Theorem 11.7), one may wonder whether requiring $\mathsf{INT\text{-}CST}$ and $\mathsf{ERR\text{-}PRE}$ is indeed necessary. A different route to achieve $\mathsf{aIND\text{-}CCFA}$ security of the encode-then-stream paradigm could be used to lift confidentiality against an active adversary directly from the $\mathsf{IND\text{-}CCFA}$ security of Ch (recall that $\mathsf{CON\text{-}CST}$ of Ch would be necessary also in this case, as explained at the beginning of Section 11.5). At first glance, one might expect lifting $\mathsf{IND\text{-}CCFA}$ to its analog in the atomic-message setting, $\mathsf{aIND\text{-}CCFA}$, should not rely on integrity of the stream-based channel. We however conjecture that such integrity is in fact necessary. Without going into details, the reason for this is that a non-integrous stream-based channel may possibly allow an attacker to modify the sent message stream in an arbitrary manner from some point on. In particular, the adversary might be able to modify the atomic-message encoding, e.g., by moving an employed end-of-message symbol to some earlier position in the message stream. Such a modification does not imply a stream-based confidentiality break, as the preceding challenge-message stream would still be suppressed. In the atomic-message sense, however, the resulting received (challenge) message is shortened, hence considered to be different and output to the adversary in the $\mathsf{aIND\text{-}CCFA}$ experiment. We therefore expect that stream-based integrity is indeed necessary to bridge the

gap from stream-based IND-CCFA to atomic-message aIND-CCFA security. This, in particular, provides a glimpse into the formal causes enabling the cookie cutter attack [BDF+14]: ultimately, atomic-message encodings need integrity protection as otherwise an adversary can restructure application messages (in this case application-layer HTTP messages) in a way that may not only violate their integrity, but also confidentiality.

In proving the theorem on integrity, the ideal target would be to build an efficient reduction that turns any successful aINT-CST adversary against the atomic-message channel $\mathsf{aCh}_{\mathsf{EtS}}$ into a successful INT-CST adversary against the streaming channel Ch. Intuitively, the reduction can simply perform the encoding and decoding operations of $\mathsf{aCh}_{\mathsf{EtS}}$ by itself and realize the streaming operations using sending and receiving oracles provided by the INT-CST experiment. The challenging part is to guarantee that any success in the aINT-CST game translates to a success in the INT-CST game. In fact, because of the different synchronization mechanisms adopted in the atomic-message setting and in the streaming setting, it is not possible to exploit every aINT-CST break against $\mathsf{aCh}_{\mathsf{EtS}}$ to violate the INT-CST property of Ch. For stream-based channels, synchronization is lost starting from the first deviating bit in the ciphertext stream. In contrast, for atomic-message channels we declare the entire ciphertext to be out-of-sync as soon as a deviation in the ciphertext sequence *or* in the message sequence is detected. Therefore, the receiving oracle in the atomic-message setting may lose synchronization 'earlier' than in the streaming setting. As we will see explicitly in the proof, CON-CST ensures that the oracle provided to the reduction from the INT-CST experiment and the one that $\mathcal{A}$ is presented with from the emulated aINT-CST experiment are consistent.

Theorem 11.16 (aINT-CST security of $\mathsf{aCh}_{\mathsf{EtS}}$)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and* $\mathsf{aCh}_{\mathsf{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be the atomic-message channel obtained from* Ch *via the encode-then-stream construction (see Construction 11.12). If* Ch *provides integrity and conciseness of ciphertext streams (*INT-CST *and* CON-CST*) then* $\mathsf{aCh}_{\mathsf{EtS}}$ *provides atomic-message integrity of ciphertexts (*aINT-CST*). Formally, for every efficient* aINT-CST *adversary* $\mathcal{A}$ *there exist an efficient* CON-CST *adversary* $\mathcal{B}$ *and* INT-CST *adversary* $\mathcal{C}$ *such that*

$$\mathsf{Adv}^{\mathsf{aINT\text{-}CST}}_{\mathsf{aCh}_{\mathsf{EtS}},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{C}}(\lambda).$$

Proof. We will show that if $\mathcal{A}$ wins the aINT-CST game against $\mathsf{aCh}_{\mathsf{EtS}}$ then we can either violate the CON-CST or the INT-CST properties of the underlying stream-based channel Ch. We start with an intuitive explanation and then give explicit reductions. Assume that $\mathcal{A}$ wins the aINT-CST game (from Figure 11.2). Then there are four possibilities for the win flag $\mathsf{win} \leftarrow 1$ to be set the first time—as we sketch below and explore in detail in the course of the proof—depending on whether $\mathsf{win} \leftarrow 1$ is set in line 23 (option #1 below) or in line 34 (options #2, #3, and #4).

#1. Synchronization has been lost before. Then C_R must be deviating from $\|\mathbf{C_S}$ and a (fully) deviating fragment causes aRecv to output some valid message.[60] As we will see, this leads to violating the INT-CST property of Ch.

#2. The stream C_R *goes ahead* of $\|\mathbf{C_S}$ (the loop of lines 25–26 terminates because $j > i$) and the exceeding part produces some valid message when processed by aRecv. This violates the INT-CST property of Ch, too.

[60] In principle, C_R could also be ahead of $\|\mathbf{C_S}$. However, in this case we can have $\mathsf{sync} = 0$ only if a valid message was already output upon processing some previous (ahead) fragment; but then $\mathsf{win} \leftarrow 1$ would have been already set. Note that once $\mathsf{aCh}_{\mathsf{EtS}}$ output the first error symbol, by construction it only outputs errors from that point on. Hence, $\mathsf{win} \leftarrow 1$ cannot be set after the first error output occurred.

#3. The stream C_R *deviates* from $\|\mathbf{C_S}$ after the first $j-1$ sent ciphertexts and messages are received entirely (the loop terminates because $j \leq i$ but $\|\mathbf{C_S}[1,\ldots,j] \not\preccurlyeq C_R$). Here the fact that $\mathcal{A}$ wins the aINT-CST game does not necessarily lead to violating the INT-CST property of Ch but, if not, it infringes its CON-CST property.

#4. The sequence $\mathbf{M_R}$ *deviates* from $\mathbf{M_S}$ after the first $j-1$ sent messages are received completely (the loop terminates because $j \leq i$ and $\|\mathbf{C_S}[1,\ldots,j] \preccurlyeq C_R$ but $\mathbf{M_S}[1,\ldots,j] \not\preccurlyeq \mathbf{M_R}[1,\ldots,j]$). Also in this case we can leverage $\mathcal{A}$'s strategy in aINT-CST to break the INT-CST security of Ch.

In the rest of the proof we first isolate the conditions causing $\mathcal{A}$ to be successful in the aINT-CST game without violating the INT-CST security of Ch—note that this can only happen for option #3. We then define a new game which penalizes $\mathcal{A}$ if the latter occurs, and bound the difference in probability between the modified game and the original one with the CON-CST advantage of an efficient adversary $\mathcal{B}$. Finally, we show that the modified game can be simulated by an efficient adversary $\mathcal{C}$ which breaks the INT-CST property whenever $\mathcal{A}$ wins the aINT-CST game.

We first set some notation. Let E^0 denote the aINT-CST experiment with the algorithms of $\mathsf{aCh}_{\mathsf{EtS}}$ plugged-in, as depicted in Figure 11.6 (ignore the framed instructions for now). Now we define a new experiment E^1 starting from E^0 by including the framed instructions (lines 9, 11, 18, 21, 33, 36–37, and 45–49). The resulting game essentially works as E^0 but it resets $\mathsf{win} \leftarrow 0$ if, although C_R contains only up to the first $j-1$ genuine ciphertexts and then deviates from $\mathbf{C_S}$, Recv produces a stream $V'_R\|\widetilde{v}$ which contains the first j genuine codewords upon processing the longest genuine prefix $\widetilde{c}$ of the first deviating query c. Informally, this change isolates the event that $\mathcal{A}$ wins the aINT-CST experiment against $\mathsf{aCh}_{\mathsf{EtS}}$ without causing a violation of the INT-CST property of Ch (i.e., a deviation from $\mathbf{C_S}$ does not translate to a deviation from the underlying message stream), and prevents $\mathcal{A}$ from winning the game in such a case. In more detail, through lines 9, 18, and 36 the new game additionally maintains a sequence $\mathbf{V_S}$ for bookkeeping the codewords input to Send as well as a string V_R for the fragments output by Recv. It makes a copy $\widetilde{\mathsf{st}'_R}$ of the current state st'_R (of the streaming algorithm) in line 21 as well as copies C'_R and V'_R of the current strings C_R and V_R in lines 33, 36, and 37 before the current query is processed by Recv. Instructions 45–47 identify the first deviating query c and perform an auxiliary call to Recv (with state $\widetilde{\mathsf{st}'_R}$, i.e., prior to processing c) on the longest genuine prefix $\widetilde{c}$ of c. Finally, instructions 48–49 detect whether processing the longest genuine prefix of V_R through Recv yields the first j codewords entirely, and, if so, set the flag bad. In what follows we denote by *bad* the event that $\mathsf{bad} \leftarrow 1$ is triggered. Finally, instruction 11 penalizes the adversary if it triggers event *bad*. Since the experiments E^0 and E^1 returns the same outcome as long as *bad* does not occur, we can bound their difference in probability by

$$\left|\mathsf{Adv}^{\mathsf{E}^0}_{\mathsf{aCh}_{\mathsf{EtS}},\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathsf{E}^1}_{\mathsf{aCh}_{\mathsf{EtS}},\mathcal{A}}(\lambda)\right| \leq \Pr[bad].$$

We now show that the occurring of event *bad* translates to a violation of the CON-CST property of the stream-based channel Ch. To this end, we build an explicit reduction $\mathcal{B}$ which simulates the game for $\mathcal{A}$ using the oracles provided by the CON-CST experiment (from Figure 11.5). The reduction $\mathcal{B}$ emulates the steps of the $\mathsf{aCh}_{\mathsf{EtS}}$ construction (cf. Figure 11.4) and uses its sending and receiving oracles from the CON-CST game to perform Send and Recv operations. However, when $\mathcal{A}$ queries the first deviating fragment c, $\mathcal{B}$ queries to its $\mathcal{O}_{\mathsf{Recv}}$ oracle the longest genuine prefix $\widetilde{c}$ of c and halts (terminating the simulation). We give the explicit code of algorithm $\mathcal{B}$ in Figure 11.7.

To see that $\mathcal{B}$ perfectly simulates the oracles of the aINT-CST experiment for $\mathcal{A}$ up to the *bad* event occurring, note that $\mathcal{B}$ essentially uses its $\mathcal{O}_{\mathsf{Send}}$ and $\mathcal{O}_{\mathsf{Recv}}$ oracles as a drop-in replacement

$\underline{\mathsf{E}^0_{\mathcal{A}}(1^\lambda),\ \mathsf{E}^1_{\mathcal{A}}(1^\lambda)\text{:}}$

$(\mathsf{st}'_{S,0}, \mathsf{st}'_{R,0}) \xleftarrow{\$} \mathsf{Init}(1^\lambda)$
$\mathsf{buf} \leftarrow \varepsilon, \mathsf{fail} \leftarrow 0$
$\mathsf{st}_S \leftarrow \mathsf{st}'_{S,0}$
$\mathsf{st}_R \leftarrow (\mathsf{st}'_{R,0}, \mathsf{buf}, \mathsf{fail})$
$\mathsf{sync} \leftarrow 1,\ \mathsf{win} \leftarrow 0$
$i \leftarrow 0$
$\mathbf{M_S} \leftarrow (),\ \mathbf{C_S} \leftarrow ()$
$\mathbf{M_R} \leftarrow (),\ C_R \leftarrow \varepsilon$
$\boxed{\mathbf{V_S} \leftarrow (),\ V_R \leftarrow \varepsilon}$
$\mathcal{A}^{\mathcal{O}_{\mathsf{Send}}, \mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$
$\boxed{\text{if } \mathsf{bad} = 1 \text{ then } \mathsf{win} \leftarrow 0}$
return win

$\underline{\mathcal{O}_{\mathsf{Send}}(m)\text{:}}$

$v \leftarrow \mathsf{Encode}(m)$
$(\mathsf{st}_S, c) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}_S, v, 1)$
$i \leftarrow i + 1$
$\mathbf{M_S}[i] \leftarrow m$
$\mathbf{C_S}[i] \leftarrow c$
$\boxed{\mathbf{V_S}[i] \leftarrow v}$
return c

$\underline{\mathcal{O}_{\mathsf{Recv}}(c)\text{:}}$

parse st_R as $(\mathsf{st}'_R, \mathsf{buf}, \mathsf{fail})$
$\boxed{\widetilde{\mathsf{st}'_R} \leftarrow \mathsf{st}'_R}$ // copy of current state
if $\mathsf{fail} = 1$ then
  $\mathbf{m} \leftarrow (\perp)$
else
  $(\mathsf{st}'_R, v) \leftarrow \mathsf{Recv}(\mathsf{st}'_R, c)$
  $\ell = \max\{|u| : u \preccurlyeq v \wedge u \in \{0,1\}^*\}$
  $v' \leftarrow v[1..\ell]$ // v' is the longest error-free prefix of v
  $\mathsf{buf} \leftarrow \mathsf{buf} \| v'$
  $(\mathbf{m}, \mathsf{buf}) \leftarrow \mathsf{Decode}(\mathsf{buf})$
  if $v' \neq v$ then
    $\mathsf{fail} \leftarrow 1, \mathbf{m} \leftarrow \mathbf{m} \| (\perp)$
  $\mathsf{st}_R \leftarrow (\mathsf{st}'_R, \mathsf{buf}, \mathsf{fail})$
$\boxed{C'_R \leftarrow C_R}$
$C_R \leftarrow C_R \| c$
$\mathbf{M_R} \leftarrow \mathbf{M_R} \| \mathbf{m}$
$\boxed{V'_R \leftarrow V_R}$
$\boxed{V_R \leftarrow V_R \| v}$
if $\mathsf{sync} = 0$ then
  if $\mathbf{m} \notin \mathcal{E}^*$ then
    $\mathsf{win} \leftarrow 1$
else if $C_R \not\preccurlyeq \| \mathbf{C_S}$ then
  $j \leftarrow 1$
  while $j \leq i$ and $\| \mathbf{C_S}[1, \ldots, j] \preccurlyeq C_R$
      and $\mathbf{M_S}[1, \ldots, j] \preccurlyeq \mathbf{M_R}$
    do $j \leftarrow j + 1$
  $\boxed{\text{if } \| \mathbf{C_S} \not\preccurlyeq C_R \text{ then}}$
    $\boxed{\widetilde{c} \leftarrow [C_R, \| \mathbf{C_S}] \,\%\, C'_R}$
    $\boxed{(\widetilde{\mathsf{st}'_R}, \widetilde{v}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}'_R}, \widetilde{c})}$
    $\boxed{\text{if } \| \mathbf{V_S}[1, \ldots, j] \preccurlyeq V'_R \| \widetilde{v} \text{ then}}$
      $\boxed{\mathsf{bad} \leftarrow 1}$
  if $j \leq i$ or $|\mathbf{M_R}| > i$ then
    $\mathsf{sync} \leftarrow 0$
  $\mathbf{m}' \leftarrow \mathbf{M_R}[j, \ldots, |\mathbf{M_R}|]$
  if $\mathbf{m}' \notin \mathcal{E}^*$ then
    $\mathsf{win} \leftarrow 1$
return $\mathbf{m}$

Figure 11.6: Security experiments $\mathsf{E}^0_{\mathcal{A}} = \mathsf{Expt}^{\mathsf{aINT\text{-}CST}}_{\mathsf{aCh}_{\mathsf{EtS}},\mathcal{A}}$ and $\mathsf{E}^1_{\mathcal{A}}$, derived from $\mathsf{E}^0_{\mathcal{A}}$ by including the framed instructions, described in the proof of Theorem 11.16.

$\mathcal{B}^{\mathcal{A},\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$:
1 $\mathsf{buf} \leftarrow \varepsilon, \mathsf{fail} \leftarrow 0$
2 $\mathsf{sync} \leftarrow 1$
3 $i \leftarrow 0$
4 $\mathbf{M_S} \leftarrow \mathbf{C_S} \leftarrow ()$
5 $\mathbf{M_R} \leftarrow (), C_R \leftarrow \varepsilon$
6 $\mathcal{A}^{\mathcal{O}^*_{\mathsf{Send}},\mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Send}}(m)$:
7 $v \leftarrow \mathsf{Encode}(m)$
8 $c \leftarrow \mathcal{O}_{\mathsf{Send}}(v, 1)$
9 $i \leftarrow i + 1$
10 $\mathbf{M_S}[i] \leftarrow m$
11 $\mathbf{C_S}[i] \leftarrow c$
12 return c to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Recv}}(c)$:
13 if $\mathsf{fail} = 1$ then return $\perp$ to $\mathcal{A}$
14 if $\mathsf{sync} = 1$ and $C_R\|c \not\preccurlyeq \|\mathbf{C_S}$ and $\|\mathbf{C_S} \not\preccurlyeq C_R\|c$ then
15 $\quad \widetilde{c} \leftarrow [C_R\|c, \|\mathbf{C_S}] \,\%\, C_R$
16 $\quad \widetilde{v} \leftarrow \mathcal{O}_{\mathsf{Recv}}(\widetilde{c})$
17 $\quad$ halt
18 $v \leftarrow \mathcal{O}_{\mathsf{Recv}}(c)$
19 $\ell = \max\{|u| : u \preccurlyeq v \wedge u \in \{0,1\}^*\}$
20 $v' \leftarrow s[1..\ell]$
21 $\mathsf{buf} \leftarrow \mathsf{buf}\|v'$
22 $(\mathbf{m}, \mathsf{buf}) \leftarrow \mathsf{Decode}(\mathsf{buf})$
23 if $v' \neq v$ then
24 $\quad \mathsf{fail} \leftarrow 1, \mathbf{m} \leftarrow \mathbf{m}\|(\perp)$
25 $C_R \leftarrow C_R\|c$
26 $\mathbf{M_R} \leftarrow \mathbf{M_R}\|\mathbf{m}$
27 return $\mathbf{m}$ to $\mathcal{A}$

Figure 11.7: Reduction $\mathcal{B}$ from the aINT-CST of $\mathsf{aCh}_{\mathsf{EtS}}$ to the CON-CST of Ch used in the proof of Theorem 11.16. Note that $\mathcal{B}$ does nothing more than emulating the construction (see Figure 11.4) until the first deviating query. Specifically, it deviates from emulating the construction only with instructions 14–17.

for the Send and Recv operations performed by the channel $\mathsf{aCh}_{\mathsf{EtS}}$, and executes the (public) encoding and decoding operations by itself.

It remains to argue that if *bad* occurs then $\mathcal{B}$ violates the CON-CST property of Ch. To this end, recall that the instructions specified in lines 45–47 identify the first deviating query c and perform an auxiliary invocation of Recv on input the longest genuine prefix $\widetilde{c}$, the latter yielding a (potentially empty) string $\widetilde{v}$. Now, in the CON-CST experiment (see Figure 11.5 for reference) for $\mathcal{B}$, once $\mathcal{B}$ has posed its last query we have that the sequence of sent ciphertexts and the strings of received ciphertext fragments, sent stream message fragments, and received stream message fragments coincide with $\mathbf{C_S}$, $C_R\|\widetilde{c}$, $\|\mathbf{V_S}$, and $V_R\|\widetilde{v}$, respectively, from the experiment E^1. By definition of *bad*, $\mathcal{B}$'s receiving queries cause Recv to output the full stream message string $\|\mathbf{V_S}[1,\ldots,j]$ although only a strict prefix of the corresponding ciphertext sequence $\mathbf{C_S}[1,\ldots,j]$ has been submitted for decryption, i.e., $\|\mathbf{C_S}[1,\ldots,j] \not\preccurlyeq C_R\|\widetilde{c}$ and $\|\mathbf{V_S}[1,\ldots,j] \preccurlyeq V_R\|\widetilde{v}$. This precisely violates the CON-CST property of Ch and hence we have

$$\Pr[bad] \leq \mathsf{Adv}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{B}}(\lambda).$$

We finally note that any $\mathcal{A}$ which is successful in experiment E^1 (from Figure 11.6) can be turned into an efficient adversary $\mathcal{C}$ that breaks the INT-CST property of Ch. Similarly to $\mathcal{B}$, algorithm $\mathcal{C}$ simulates the $\mathsf{aCh}_{\mathsf{EtS}}$ operations using the oracles provided by the INT-CST experiment to perform Send and Recv on message fragments. In contrast to $\mathcal{B}$ (which halts when $\mathcal{A}$ poses the first deviating query), $\mathcal{C}$'s simulation goes on until $\mathcal{A}$ stops.

As for $\mathcal{B}$, it is immediate to see that $\mathcal{C}$ perfectly emulates the oracles for $\mathcal{A}$. The more challenging part of the proof is to show that if $\mathcal{A}$ is successful, so is $\mathcal{C}$. More specifically, we show that if $\mathcal{A}$ wins, by triggering lines 40 or 54 in game E^1 (from Figure 11.6) but not the *bad* event (lines 49 and 11), then $\mathcal{C}$ wins through triggering lines 18 or 35 in the INT-CST game (from Figure 10.5 on page 158).

In the rest of the proof we will use the properties of the $\mathsf{aCh}_{\mathsf{EtS}}$ construction. First, aSend always invokes Send with flush flag set to $f = 1$: this induces a natural correspondence between $\mathbf{M_S}$, $\mathbf{V_S}$ and $\mathbf{C_S}$. Second, upon processing a receiving query c and the corresponding string v output by Recv in experiment E^1, the sequence $\mathbf{M_R}$ is updated by first appending the

message vector $\mathbf{m}$ obtained by decoding the *valid* part v' of fragment v and then, if v contains any error, by also appending the symbol $\perp$ (see lines 26–29 in Figure 11.6). Moreover, once the first error occurs, $\mathbf{M_R}$ is only further augmented with errors (see line 23), so $\mathcal{A}$ cannot win after this point.

We already saw that there are essentially four possibilities for $\mathcal{A}$ to win the aINT-CST experiment (and so in game E^1). We next show that, assuming that $\mathcal{A}$ is successful, in each of these cases $\mathcal{C}$ violates the INT-CST property of Ch. Let c denote the receiving query (input) that causes $\mathsf{win} \leftarrow 1$ to be set the first time.

We consider first the case in which $\mathsf{sync} \leftarrow 0$ is set with some query prior to c ($\mathcal{A}$ wins by triggering instruction 40 in Figure 11.6).

Case #1. Synchronization has been lost before. Then C_R must be deviating from $\|\mathbf{C_S}$ (see footnote 60 on page 185). Note that for $\mathcal{A}$ to win, aRecv on input the (fully deviating) fragment c must output some message sequence $\mathbf{m} \neq (\perp)$. That is, if buf denotes the buffer kept by aRecv prior to processing c, we have that Recv on input c returns a string v such that $\mathsf{Decode}(\mathsf{buf}\|v') = (\mathbf{m}, \mathsf{buf}')$, where v' is the longest error-free prefix of v. In other words, processing c augments the buffer to contain (at least) a full codeword v^* that was not completed before receiving c (otherwise $\mathsf{win} \leftarrow 1$ would have been set with some previous query).

In more detail, let c'_1 denote the first deviating query, let $c'_2, \ldots, c'_\ell$ be the subsequent decryption queries prior to c, and let $v'_1, \ldots, v'_\ell$ be the output of Recv on input these queries. Since $\mathsf{win} \leftarrow 1$ is only set when $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}$ on c, the sequence of messages received after processing each of $c'_1, \ldots, c'_\ell$ must be a prefix of $\mathbf{M_S}$, i.e., $\mathbf{M_R} = \mathbf{M_S}[1, \ldots, k]$ for some $1 \leq k < i$, and moreover $v'_1, \ldots, v'_\ell \in \{0,1\}^*$ do not contain an error symbol. Prior to receiving c we must have that $\|\mathbf{V_S}[1, \ldots, k] \preccurlyeq V_R$ and the remainder $V_R \,\%\, \|\mathbf{V_S}[1, \ldots, k] = v''_1\|v'_2\| \cdots \|v'_\ell$, where v''_1 is a suffix of v'_1 (some genuine prefix of v'_1 may complete the codeword $\mathbf{V_S}[k]$), does not contain any full codeword (again, otherwise $\mathsf{win} \leftarrow 1$ would be set earlier). Only when c is processed and its output v' appended to the current buffer can we isolate a full codeword v^*, i.e., $v^* \preccurlyeq \mathsf{buf} = v''_1\| \cdots \|v'_\ell\|v'$ such that $\mathsf{Decode}(v^*) = (\mathbf{M_R}[k+1], \varepsilon)$. In particular, $v^* \notin \mathcal{E}^*$. Thus, some among the deviating queries $c'_1, \ldots, c'_\ell, c$ that $\mathcal{C}$ forwards to its own receiving oracle will make $\mathcal{C}$ win in the INT-CST experiment by triggering line 18 or line 35 in Figure 10.5.

The next three cases are those induced by the clauses defining the predicate of line 25 in the aINT-CST experiment (Figure 11.2). Here $\mathsf{win} \leftarrow 1$ and $\mathsf{sync} \leftarrow 0$ are set within the same $\mathcal{O}_{\mathsf{Recv}}$ call in E^1 (in lines 51 and 54). Note that for the aINT-CST experiment to enter the loop of lines 25–26 the current query c must cause $C_R \not\preccurlyeq \|\mathbf{C_S}$. The three cases to consider are the following.

Case #2. C_R goes *ahead* of $\|\mathbf{C_S}$ (the loop terminates because $j > i$). Then after c is processed we have $j = i + 1$, $\|\mathbf{C_S} \prec C_R$ and $\mathbf{M_S} \preccurlyeq \mathbf{M_R}$. Assuming that $\mathcal{A}$ wins the game we must additionally have $\mathbf{M_S}[1, \ldots, i] \prec \mathbf{M_R}$ and in particular $\mathbf{M_R}[j] \neq \perp$. It follows from the conditions above that $V_R = \|\mathbf{V_S}[1, \ldots, i]\|v^*\|s$ such that $\mathsf{Decode}(v^*) = (\mathbf{M_R}[j], \varepsilon)$.[61] That is, Recv outputs on input c a string $v'\|v^*\|s \notin \mathcal{E}^*$, where v' might complete previous fragments. Hence, this triggers instruction 35 (in Figure 10.5), making $\mathcal{C}$ violate the INT-CST property of Ch.

[61] In fact, the implication $\mathbf{M_S} \preccurlyeq \mathbf{M_R} \Longrightarrow \|\mathbf{V_S} \preccurlyeq V_R$ holds under the assumption that only codewords are decoded to valid messages (i.e., there exists no string $x \in \{0,1\}^* \setminus V$ such that $\mathsf{Decode}(x) = (\mathbf{m}, s)$ with $\mathbf{m} \in \mathcal{M}^*$). For the argument to go through, however, we do not need such an assumption, as having $\|\mathbf{V_S} \not\preccurlyeq V_R$ here would immediately lead to a violation of the INT-CST property of Ch. A similar argument also applies to cases #3 and #4.

Case #3. The stream C_R *deviates* from $\|\mathbf{C_S}$ after the first $j-1$ sent ciphertexts and messages are received entirely (the loop terminates because $j \le i$ but $\|\mathbf{C_S}[1,\ldots,j] \not\preccurlyeq C_R$). In this case we have $\|\mathbf{C_S}[1,\ldots,j-1] \prec C_R$ and $\mathbf{M_S}[1,\ldots,j-1] \preccurlyeq \mathbf{M_R}$, but $\|\mathbf{C_S}[1,\ldots,j] \not\preccurlyeq C_R$. Again assuming that $\mathsf{win} \leftarrow 1$ is set with the current query we have $\mathbf{M_S}[1,\ldots,j-1] \prec \mathbf{M_R}$ and $\mathbf{M_R}[j] \neq \bot$. Then $V_R = \|\mathbf{V_S}[1,\ldots,j-1]\|v^*\|s$ with $v^* \in \{0,1\}^*$, $s \in (\{0,1\} \cup \mathcal{E})^*$ and $\mathsf{Decode}(v^*) = (\mathbf{M_R}[j], \varepsilon)$. This means that on processing the current (deviating) query the stream-based algorithm Recv returns (beyond a potential genuine prefix) a non-error output $v = v^*\|s$, i.e., $v \notin \mathcal{E}^*$. Now, if $\mathbf{M_R}[j] \neq \mathbf{M_S}[j]$ we have $v^* \neq \mathbf{V_S}[j]$ and, because of the prefix-freeness of the set of codewords, that $v^* \not\preccurlyeq \mathbf{V_S}[j]$, which is an evident violation of the $\mathsf{INT\text{-}CST}$ property. The case $\mathbf{M_R}[j] = \mathbf{M_S}[j]$ requires more care. Indeed, since $v^* = \mathbf{V_S}[j]$, we cannot directly argue that having $v^* \in \{0,1\}^*$ violates the $\mathsf{INT\text{-}CST}$ property of Ch, as v^* may be also output by Recv when processing the *genuine* prefix $\widetilde{c}$ of c. However, the latter would imply $v^* = \mathbf{V_S}[j] \preccurlyeq \widetilde{v}$, hence triggering the *bad* event (instructions 49 and 11 in Figure 11.6), meaning that $\mathcal{A}$ loses the game, against our assumption. That is, Recv only outputs the full v^* after processing the deviating part of c, hence violating the $\mathsf{INT\text{-}CST}$ security of Ch.

Case #4. The sequence $\mathbf{M_R}$ *deviates* from $\mathbf{M_S}$ after the first $j-1$ sent messages are received completely (the loop terminates because $j \le i$ and $\|\mathbf{C_S}[1,\ldots,j] \preccurlyeq C_R$ but $\mathbf{M_S}[1,\ldots,j] \not\preccurlyeq \mathbf{M_R}[1,\ldots,j]$). As $\mathsf{win} \leftarrow 1$ is set with the current query we have $\mathbf{M_S}[1,\ldots,j-1] \preccurlyeq \mathbf{M_R}$, $\mathbf{M_S}[1,\ldots,j] \not\preccurlyeq \mathbf{M_R}$ and $\mathbf{M_R}[j] \neq \bot$. Then $V_R = \|\mathbf{V_S}[1,\ldots,j-1]\|v^*\|s$ for some $v^* \in \{0,1\}^*$ and $s \in (\{0,1\} \cup \mathcal{E})^*$ such that $\mathsf{Decode}(v^*) = (\mathbf{M_R}[j], \varepsilon)$, where $v^* \neq \mathbf{V_S}[j]$ and, for the prefix-freenes, $v^* \not\preccurlyeq \mathbf{V_S}[j]$, which again violates the $\mathsf{INT\text{-}CST}$ of Ch.

The analysis above proves that

$$\mathsf{Adv}^{\mathsf{E}^1}_{\mathsf{aCh_{EtS}},\mathcal{A}}(\lambda) \le \mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{A}}(\lambda).$$

Overall, we hence obtain the final bound

$$\mathsf{Adv}^{\mathsf{aINT\text{-}CST}}_{\mathsf{aCh_{EtS}},\mathcal{A}}(\lambda) \le \mathsf{Adv}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{C}}(\lambda). \qquad \square$$

We saw in Theorem 11.16 that the encode-then-stream paradigm allows us to leverage integrity and conciseness of ciphertext streams for stream-based channels ($\mathsf{INT\text{-}CST}$ and $\mathsf{CON\text{-}CST}$) to integrity of ciphertexts for atomic-message channels ($\mathsf{aINT\text{-}CST}$). In the following theorems we prove a similar result for confidentiality. First, we show in Theorem 11.17 that the encode-then-stream paradigm lifts confidentiality against chosen plaintext-fragment attacks ($\mathsf{IND\text{-}CPFA}$) to confidentiality against (atomic) chosen-plaintext attacks ($\mathsf{aIND\text{-}CPA}$). Then, we prove in Theorem 11.18 that $\mathsf{aCh_{EtS}}$ maintains (in the atomic setting) the error predictability of the underlying stream-based channel. Finally, as a corollary of the above results together with Theorem 11.7, we conclude that $\mathsf{CON\text{-}CST}$, $\mathsf{INT\text{-}CST}$, $\mathsf{IND\text{-}CPFA}$, and $\mathsf{ERR\text{-}PRE}$ of Ch imply $\mathsf{aIND\text{-}CCFA}$ security of $\mathsf{aCh_{EtS}}$.

Theorem 11.17 ($\mathsf{aIND\text{-}CPA}$ security of $\mathsf{aCh_{EtS}}$)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and* $\mathsf{aCh_{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be the atomic-message channel obtained from* Ch *via the encode-then-stream construction (see Construction 11.12). If* Ch *provides indistinguishability under chosen plaintext-fragment attacks (*$\mathsf{IND\text{-}CPFA}$*) then* $\mathsf{aCh_{EtS}}$ *provides indistinguishability under chosen-plaintext attacks in the atomic-message setting (*$\mathsf{aIND\text{-}CPA}$*). Formally, for every efficient* $\mathsf{aIND\text{-}CPA}$ *adversary* $\mathcal{A}$ *there exists an efficient* $\mathsf{IND\text{-}CPFA}$ *adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathsf{aIND\text{-}CPA}}_{\mathsf{aCh_{EtS}},\mathcal{A}} \le \mathsf{Adv}^{\mathsf{IND\text{-}CPFA}}_{\mathsf{Ch},\mathcal{B}}.$$

Proof. Given a left-or-right oracle from the $\mathsf{IND\text{-}CPFA}$ experiment against Ch it is immediate to simulate the $\mathsf{aIND\text{-}CPA}$ experiment. Whenever $\mathcal{A}$ queries (m_0, m_1), algorithm $\mathcal{B}$ computes $v_0 =$

$\mathsf{Encode}(m_0)$, $v_1 = \mathsf{Encode}(m_1)$, queries$(v_0, v_1, 1)$ to its own oracle $\mathcal{O}_{\mathsf{LoR}}$, and gives the result c to $\mathcal{A}$. Observe that $|v_0| = |v_1|$ due to the length-regularity of ES, so the queries made by $\mathcal{B}$ to its oracle are valid. As soon as $\mathcal{A}$ stops and returns a bit b', so does $\mathcal{B}$. Hence, analogously to the reductions described in the proof of Theorem 11.16, $\mathcal{B}$ perfectly emulates the oracle for $\mathcal{A}$, executing the streaming Send operation via its oracle. Thus, $\mathcal{B}$ has the same advantage as $\mathcal{A}$. □

Theorem 11.18 (aERR-PRE security of $\mathsf{aCh}_{\mathsf{EtS}}$)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and* $\mathsf{aCh}_{\mathsf{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be the atomic-message channel obtained from* Ch *via the encode-then-stream construction (see Construction 11.12). If* Ch *provides error predictability (*ERR-PRE*) with respect to some efficient predictor* Pred *then there exists an efficient predictor* Pred' *(described in the proof) such that* $\mathsf{aCh}_{\mathsf{EtS}}$ *is error predictable (*aERR-PRE*) with respect to* Pred'*. Formally, for every efficient* aERR-PRE *adversary* $\mathcal{A}$ *there exist an efficient* ERR-PRE *adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathsf{aERR\text{-}PRE}}_{\mathsf{aCh}_{\mathsf{EtS}},\mathsf{Pred}',\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{B}}.$$

Proof. Let Pred' be an algorithm that on input $\mathbf{C_S}$, C_R, and c (as described in the experiment from Figure 11.3) invokes Pred as a subroutine on input $\|\mathbf{C_S}$, C_R, and c. Then, depending on the output e of Pred, Pred' returns an empty vector () if $e = \varepsilon$ is the empty string, otherwise it returns $(\bot)$.

To see that Pred' is a valid error predictor for $\mathsf{aCh}_{\mathsf{EtS}}$, notice first that by construction algorithm aRecv always outputs an element $\mathbf{m}$ in $\mathcal{M}^* \cup (\mathcal{M}^* \times \bot)$, i.e., a vector of messages possibly followed by the error symbol $\bot$. Thus, for the projection $\langle \mathbf{m} \rangle_{\mathcal{E}}$ there are only two possibilities: it is either the empty vector () or the singleton $(\bot)$. Now let $\mathcal{A}$ be a successful adversary in the aERR-PRE experiment (from Figure 11.3) against the error predictor Pred'. Then we can build an algorithm $\mathcal{B}$, to be run in the ERR-PRE experiment (from Figure 10.6) against Pred, similarly to the reductions described in Theorems 11.16 and 11.17, i.e., by letting $\mathcal{B}$ emulate the oracles for $\mathcal{A}$ by performing the public encoding and decoding operations of $\mathsf{aCh}_{\mathsf{EtS}}$ and using its sending and receiving oracles for the streaming operations.

There are only two possibilities for $\mathcal{A}$ to be successful (in line 13 of Figure 11.3): either (i) Pred' returns () while aRecv outputs an error, thus $\langle \mathbf{m} \rangle_{\mathcal{E}} = (\bot)$, or (ii) Pred' returns $(\bot)$ but aRecv only produces valid messages, thus $\langle \mathbf{m} \rangle_{\mathcal{E}} = ()$. Observe that by construction Pred' returns an empty vector if and only if Pred returns an empty string. Similarly, aRecv outputs $\bot$ if and only if Recv also returns some error symbols. We hence deduce that, in the first case, Recv on input c returns error symbols but Pred erroneously predicts that no error occurred, causing the execution of line 11. The second case is analogous: Recv on input c only produces valid message bits, while Pred falsely predicts errors. In either case, $\mathcal{B}$ wins by causing the execution of line 11 in Figure 10.6. □

Corollary 11.19 (aIND-CCFA security of $\mathsf{aCh}_{\mathsf{EtS}}$)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and* $\mathsf{aCh}_{\mathsf{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be the atomic-message channel obtained from* Ch *via the encode-then-stream construction (see Construction 11.12). If* Ch *provides integrity and conciseness of ciphertext streams (*INT-CST *and* CON-CST*), error predictability (*ERR-PRE*), and indistinguishability under chosen plaintext-fragment attacks (*IND-CPFA*), then* $\mathsf{aCh}_{\mathsf{EtS}}$ *provides indistinguishability under chosen-ciphertext attacks in the atomic-message setting (*aIND-CCFA*).*

11.5.3 Secure Instantiation from the AEAD-based Streaming Channel

The results of the previous section establish sufficient security requirements on the underlying stream-based channel to obtain a secure atomic-message channel via the encode-then-stream paradigm. One of these requirements is conciseness of ciphertext streams (CON-CST), which

essentially says that ciphertext fragments produced by the streaming sending algorithm contain no redundant bits that can be chopped or modified without altering the underlying message fragment. It is not difficult to achieve this property. Indeed, as we show in the next theorem, our AEAD-based streaming channel $\mathsf{Ch}_{\mathsf{AEAD}}$ from Construction 10.12 in Section 10.4 provides conciseness of ciphertext streams. This result holds unconditionally in an information-theoretic sense; even an unbounded adversary cannot violate CON-CST of $\mathsf{Ch}_{\mathsf{AEAD}}$.

Theorem 11.20 (CON-CST of $\mathsf{Ch}_{\mathsf{AEAD}}$)**.** *The stream-based channel* $\mathsf{Ch}_{\mathsf{AEAD}}$ *from Construction 10.12 in Section 10.4 unconditionally provides conciseness of ciphertext streams (*CON-CST*). Formally, for any (even unbounded)* CON-CST *adversary* $\mathcal{A}$ *against* $\mathsf{Ch}_{\mathsf{AEAD}}$ *it holds that*

$$\mathsf{Adv}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch}_{\mathsf{AEAD}},\mathcal{A}} = 0.$$

Proof. Observe that a genuine ciphertext fragment c produced by Send on input some message fragment $m \in \{0,1\}^*$ is, by construction, the concatenation of a number of blocks $B_1, B_2, \ldots$, each containing an AEAD ciphertext c'_i preceded by (the bit representation of) its length len_i. Importantly, Recv identifies these blocks in the received ciphertext stream and only upon receiving a *full* block $B_i = len_i \| c'_i$ does it invoke the AEAD decryption algorithm on ciphertext c'_i. Thus, it is impossible to violate conciseness of ciphertexts. Indeed, recall from the CON-CST game (see Figure 11.5) that in order to win the adversary can only submit to $\mathcal{O}_{\mathsf{Recv}}$ a strict prefix of the genuine ciphertext stream. However, in the case of $\mathsf{Ch}_{\mathsf{AEAD}}$ not even an unbounded adversary can make Recv output one of the sent message fragments *entirely* by submitting a truncation of the corresponding ciphertext fragment: chopping the ciphertext prevents the AEAD decryption from being invoked on some block B_i and, consequently, causes a truncation of the genuine message stream. □

Remark 11.21. The TLS record protocol in both versions 1.2 and 1.3 [DR08, Res18] also provides conciseness of its ciphertext stream when using an AEAD scheme and omitting the option to send zero-length message fragments. As in our construction, TLS records are data structures clearly marked-out via a length header, forming blocks within the ciphertext fragments output by the sender. Hence, for reasons similar to those in Theorem 11.20, even an unbounded adversary cannot force entire message fragments to be output when receiving truncated ciphertext fragments.

Theorem 11.20 together with the already established INT-CST and IND-CCFA security of $\mathsf{Ch}_{\mathsf{AEAD}}$ under the mild assumption that the authenticated encryption scheme with associated data AEAD provides INT-CTXT and IND-CPA security suggests that $\mathsf{Ch}_{\mathsf{AEAD}}$ is a 'good' stream-based channel to start with for building secure atomic-message channels using the encode-then-stream transform. Moreover, the atomic-message channel construction $\mathsf{aCh}_{\mathsf{EtS}}$ applied to $\mathsf{Ch}_{\mathsf{AEAD}}$ yields a secure atomic-message channel from AEAD.

Corollary 11.22 (Atomic-message channels from AEAD)**.** *Let* AEAD *be an authenticated encryption scheme with associated data, let* $\mathsf{Ch}_{\mathsf{AEAD}}$ *be the streaming channel obtained from* AEAD *via Construction 10.12 in Section 10.4, and let* $\mathsf{aCh}_{\mathsf{EtS}}$ *be the atomic-message channel construction from Construction 11.12 applied to* $\mathsf{Ch}_{\mathsf{AEAD}}$*. If* AEAD *provides integrity of ciphertexts (*INT-CTXT*) and indistinguishability under chosen-plaintext attacks (*IND-CPA*) then* $\mathsf{aCh}_{\mathsf{EtS}}$ *provides atomic integrity of ciphertext streams (*aINT-CST*) and atomic indistinguishability under chosen ciphertext-fragment attacks (*aIND-CCFA*).*

Chapter 12

Multi-key Channels

Summary. In this chapter we present our framework for multi-key channels that captures secure channel designs in which a key-updating mechanism allows to deploy a sequence of multiple encryption keys instead of a single, fixed key. We begin by presenting a modular specification and security notions for confidentiality and integrity of multi-key channels, integrating two advanced security aspects aimed at by key updating, forward security and phase-key insulation. We study relations within the resulting hierarchy of security notions and show that the lower-end notions naturally connect to the existing notions of stateful encryption established for single-key channels. We finally instantiate the strongest security notions in our model with a construction based on authenticated encryption with associated data and a pseudorandom function which reflects aspects of the TLS 1.3 record protocol design and hence enables a comparative discussion. The results in this chapter are based on a work published at CRYPTO 2017 [GM17].

12.1 Introduction

In all cryptographic models of secure channels established so far, security originates from a single, symmetric key shared between the two endpoints of the channel (see Chapters 1 and 9 for an overview). The upcoming version of the TLS protocol, TLS 1.3 [Res18], however deviates from this paradigm and instead deploys a series of multiple keys in a sequence of *phases*. The TLS 1.3 channel (the so-called record protocol) as usual begins with deriving an initial key for encryption and decryption of messages. As a novel component, both parties are further able to trigger key updates, leading to a key switch and new phase according to a pre-defined schedule while maintaining the channel's operation. One particular motivation for this approach is that long-lived TLS connections may exhaust the cryptographic limits of some algorithms on how much data can be safely encrypted under a single key (cf. [Res18, Section 5.5], [LP16]).

A more general, major reason for refreshing the key used in a secure channel and specifically TLS 1.3 is *forward security*, a notion primarily known from and well-established in the context of key exchange protocols [Gün90, DVOW92, CK01] (see also Part I of this thesis). When using the same key throughout the lifetime of a channel, an attacker that learns this key (e.g., through cryptanalysis or even temporary break-in into the system) immediately compromises the confidentiality of previous and the integrity of future communication. In contrast, forward security demands that even if key material is leaked at some point, previous communication remains secure. Forward-secure symmetric encryption in the non-stateful setting is considered understood and in particular can be built from forward-secure pseudorandom bit generators [BY03] or, more generally, through re-keying [AB00]. In the context of secure channels, a formal treatment of forward security is however lacking so far.

Beyond forward security, a second security property arises for secure channels (in particular in the design of TLS 1.3) which we refer to as *phase-key insulation*. While forward security targets a full compromise (and prior security), phase-key insulation is concerned with the *temporary* compromise of a channel in the form of leaking the key used in a certain *phase* (i.e., time period), but not in others. Such temporary compromise might, e.g., result from differing strengths of key material used to derive some of the phase keys (as is the case for keys established in the TLS 1.3 key exchange [KW16, DFGS15a, FG17], see also Chapters 6 and 7) or from storing the currently active key in less secure memory for efficiency reasons. A secure channel with phase-key insulation should then uphold confidentiality and integrity in uncompromised phases, even if the key of prior or later phases is revealed. Moreover, security should be retained even if the attacker learned a phase's key while that phase was still active. Our notion of phase-key insulation is similar in spirit to (and hence borrows its name from) the notion of key insulation introduced in the public-key setting [DKXY02, DKXY03] and also transferred to (non-stateful) symmetric encryption [DLXY12].

As we will see, phase-key insulation complements and is orthogonal to the notion of forward security, which is only concerned with a-posteriori leakage of keys. Requiring it furthermore introduces new pitfalls in the design of secure channels. For example, the initial draft design of the TLS 1.3 record protocol with key updates enabled truncation attacks in non-compromised phases that would go unnoticed during the further execution of the protocol, as Fournet and the miTLS [miT] team discovered [Fou15]. We hence consider it being crucial to establish a formal understanding of channels using multiple keys in order to provide means for evaluating the provable security guarantees of proposed protocols.

Multi-key channels. In this chapter we study channels that employ a sequence of multiple keys. To this end, we introduce a formalization of such *multi-key channels* and set up an according framework of game-based security notions. We then analyze the relations between our security notions as well as connections to the established notions for stateful encryption. Finally, we provide a generic construction of a provably secure multi-key channel achieving the strongest notions in our framework.

Following the game-based tradition in modeling channels, our formalism builds upon and extends that of Bellare, Kohno, and Namprempre [BKN04] (cf. Section 9.3) and Bellare and Yee [BY03]. More specifically, our notion of multi-key channels augments that of regular stateful encryption in three aspects. Obviously, we first of all consider a sequence of keys to be used for encryption and decryption. Secondly, switches between these keys are initiated through a specific key-update algorithm which makes the channel proceed from one phase to the next. Lastly, we separate two hierarchies of keys by additionally considering a level of master secret keys which, also evolving over time, are used to derive the channel key for each phase. As we will discuss, this carefully crafted syntax and key hierarchy in particular allows us to closely model the key schedule of the TLS 1.3 record protocol draft [Res18].

We then define security of multi-key channels via a framework of notions. Beyond capturing the classical requirements of confidentiality and integrity, our notions modularly integrate the advanced security properties of forward security and phase-key insulation arising in the context of multi-key channels. The core technical challenge here is to appropriately capture the desired security properties while excluding trivial attacks in the stateful multi-key setting. We furthermore modularize the adversary's capability to proceed a channel to a next phase through key updates. Thereby, our framework elegantly also captures the single-key variants of our security notions, i.e., the cases where a multi-key channel only operates in a single phase.

Our single-key security notions enable us to provide a formal link to the established stateful-encryption notions for regular channels. We show that analogous notions in both models are

essentially equivalent (modulo the differences in syntax) by providing natural, generic transforms between each pair of corresponding confidentiality and integrity notions. Furthermore, we establish separations that give rise to a hierarchy of our security notions and in particular establish forward security and phase-key insulation as independent security properties. To complete the picture of relations, we also translate the classical composition result for symmetric encryption by Bellare and Namprempre [BN00] to the setting of multi-key channels, showing that chosen-plaintext confidentiality combined with ciphertext integrity implies the stronger chosen-ciphertext notion of confidentiality.

Finally, we instantiate our model by providing a construction of a multi-key channel from a nonce-based authenticated encryption with associated data (AEAD) scheme and a pseudorandom function. To ensure both forward security and phase-key insulation, we match suitable techniques established for forward-secure key generation and for ensuring causal integrity. Leveraging our composition theorem, we then prove that our construction meets our strongest confidentiality and integrity notions for multi-key channels. Coming back to the initial motivation from real-world protocol design, we compare our construction with the draft design of the TLS 1.3 record protocol.

12.2 Syntax and Functionality of Multi-key Channels

We begin with defining the syntax and correctness of multi-key channels, focusing on their functionality in this section; we will treat their security in Section 12.3. In Figure 12.1 we exemplify the operations of a multi-key channel and already hint at their expected security.

Like a regular, single-key channel (abstractly modeled as stateful encryption [BKN04], cf. Section 9.3), a multi-key channel is used by a sender to transform a sequence of messages m_1, $m_2, \ldots \in \{0,1\}^*$ into a corresponding sequence of ciphertexts $c_1, c_2, \ldots \in \{0,1\}^*$ using a sending algorithm Send. The receiver then sequentially uses a corresponding Recv algorithm on each transmitted ciphertext to recover the sent message sequence.

In contrast to the preceding Chapters 10 and 11, we model the sending and receiving algorithms again as processing *atomic* messages and ciphertext without fragmentation, as for the original notion of stateful encryption. Without question, a comprehensive game-based security treatment of, e.g., the TLS 1.3 record protocol would in the end need to capture both its streaming behavior (cf. Chapter 10) and multi-key properties. As we will see, capturing the effects of key updating however introduces a significant amount of complexity already in the simpler setting with atomic messages and ciphertexts. We therefore choose to restrict ourselves to this setting for the first exploration of multi-key channels and leave combining multi-key and streaming channels as an interesting avenue for future work.

In addition to regular channels, in a multi-key channel both sender and receiver can decide to update their keys used for sending and receiving, thereby switching to the next *phase* of the channel. In our model, we consider a two-level hierarchy for key derivation. On the first level, the whole multi-key channel is bootstrapped from a single, initial *master secret key* generated upon initialization of the channel. Master secret keys are furthermore evolved when switching to the next phase, following a deterministic key schedule to derive the master secret key msk_{t+1} for phase $t+1$ from the master secret key msk_t of the previous phase. On the second level, the actual *phase key* K_t used in the channel for sending and receiving messages in a phase t is derived (again deterministically) from that phase's master secret key msk_t. In order to make the current phase key used for sending and receiving explicit we provide it as an distinguished input to the Send and Recv algorithms beyond the general sending and receiving state.

Although Figure 12.1 depicts only a single key schedule with the phase keys forwarded to both the Send and Recv algorithms of that phase, in a real execution of the channel, the key

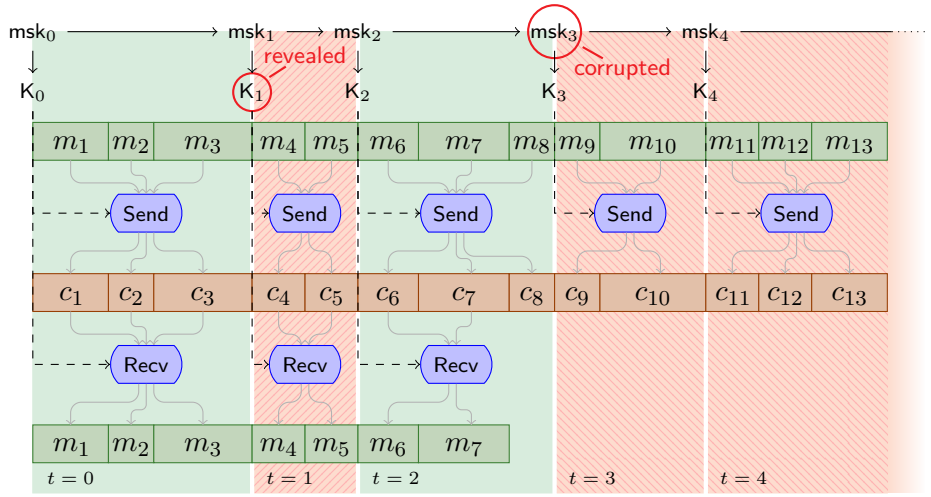


Figure 12.1: Illustration of the behavior of a multi-key channel (cf. Definition 12.1). The beginning of a new phase t is indicated by the derivation of a phase key K_t from the corresponding master secret key msk_t. The phase key K_t is then used to send and receive in-order messages, resp. ciphertexts, via algorithms Send and Recv in this phase.
In this example, the phase key K_1 of phase $t = 1$ is revealed and the master secret key msk_3 of phase $t = 3$ is corrupted. The affected phases 1, resp. 3, 4, and following are marked in hatched-pattern red (with lines towards top right for the effects of the revealed K_1 and towards bottom right for the effects of the corrupted msk_3). For security (cf. Section 12.3), a forward-secure and phase-key–insulated multi-key channel is demanded to provide security in the non-affected phases 0 and 2, marked by non-hatched green areas.

updates and derivations are invoked independently on the sending and receiving side. For correct functionality, key updates need to be aligned in order to process sent and received ciphertexts under matching keys on both sides. In practice, key update notifications may be either delivered alongside of the messages transmitted in the channel (and hence potentially authenticated) or in an out-of-band manner, e.g., via a separate control channel, and with their position in the channel's ciphertext sequence not being explicitly authenticated.[62] Alternatively, key updates may also be scheduled deterministically, e.g., after a certain maximum number of messages have been sent within one phase. In our abstraction of multi-key channels, we do not rely on the authenticity of the key-update signaling (in particular, we will later allow adversaries to tamper with the timing of key updates) but leave it up to the multi-key channel to ensure their correct position with respect to the transmitted ciphertexts.

We now define the syntax and correctness of multi-key channels capturing the given intuition.

Definition 12.1 (Syntax of multi-key channels)**.** *A* multi-key channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv}, \mathsf{Update})$ *with associated sending and receiving state space* $\mathcal{S}_S$ *resp.* $\mathcal{S}_R$*, master secret key space* $\mathcal{MSK}$*, phase key space* $\mathcal{K}$*, error space* $\mathcal{E}$ *with* $\mathcal{E} \cap \{0,1\}^* = \emptyset$*, and maximum number* $\mathsf{maxmsg} \in \mathbb{N} \cup \{\infty\}$ *of messages supported per phase consists of four efficient algorithms defined as follows.*

- $\mathsf{Init}(1^\lambda) \xrightarrow{\$} (\mathsf{msk}_0, \mathsf{K}_0, \mathsf{st}_{S,0}, \mathsf{st}_{R,0})$*. This probabilistic algorithm is composed of three sub-algorithms:*
  - $\mathsf{MasterKeyGen}(1^\lambda) \xrightarrow{\$} \mathsf{msk}_0$*. On input a security parameter* 1^λ*, this probabilistic algorithm outputs an initial master secret key* $\mathsf{msk}_0 \in \mathcal{MSK}$*.*

[62] In the context of TLS 1.3, for example, both variants have been discussed. The current draft design [Res18] specifies that key update notifications are transmitted (and authenticated) within the data channel.

  - $\mathsf{KeyDerive}(\mathsf{msk}) \to \mathsf{K}$. *On input a master secret key* msk*, this deterministic algorithm outputs a phase key* $\mathsf{K} \in \mathcal{K}$*. The initial phase key is derived as* $\mathsf{K}_0 \leftarrow \mathsf{KeyDerive}(\mathsf{msk}_0)$.
  - $\mathsf{StateGen}(1^\lambda) \to (\mathsf{st}_{S,0}, \mathsf{st}_{R,0})$. *On input a security parameter* 1^λ*, this deterministic algorithm outputs initial sending and receiving states* $\mathsf{st}_{S,0} \in \mathcal{S}_S$ *resp.* $\mathsf{st}_{R,0} \in \mathcal{S}_R$.

- $\mathsf{Send}(\mathsf{st}_{S,t}, \mathsf{K}_t, m) \xrightarrow{\$} (\mathsf{st}'_{S,t}, c)$. *On input a sending state* $\mathsf{st}_{S,t} \in \mathcal{S}_S$*, a phase key* $\mathsf{K}_t \in \mathcal{K}$*, and a message* $m \in \{0,1\}^*$*, this (possibly) probabilistic algorithm outputs an updated state* $\mathsf{st}'_{S,t} \in \mathcal{S}_S$ *and a ciphertext (or error symbol)* $c \in \{0,1\}^* \cup \mathcal{E}$.

- $\mathsf{Recv}(\mathsf{st}_{R,t}, \mathsf{K}_t, c) \to (\mathsf{st}'_{R,t}, m)$. *On input a receiving state* $\mathsf{st}_{R,t} \in \mathcal{S}_R$*, a phase key* $\mathsf{K}_t \in \mathcal{K}$*, and a ciphertext* $c \in \{0,1\}^*$*, this deterministic algorithm outputs an updated state* $\mathsf{st}'_{R,t} \in \mathcal{S}_R$ *and a message (or error symbol)* $m \in \{0,1\}^* \cup \mathcal{E}$.

- $\mathsf{Update}(\mathsf{msk}_t, \mathsf{st}_{S,t}/\mathsf{st}_{R,t}) \to (\mathsf{msk}_{t+1}, \mathsf{K}_{t+1}, \mathsf{st}_{S,t+1}/\mathsf{st}_{R,t+1})$. *This deterministic algorithm is composed of the following two sub-algorithms:*

  - $\mathsf{MasterKeyUp}(\mathsf{msk}_t) \to \mathsf{msk}_{t+1}$. *On input a master secret key* $\mathsf{msk}_t \in \mathcal{MSK}$*, this deterministic algorithm outputs a master secret key* $\mathsf{msk}_{t+1} \in \mathcal{MSK}$ *for the next phase.*
  - $\mathsf{StateUp}(\mathsf{st}_{S,t}/\mathsf{st}_{R,t}) \to \mathsf{st}_{S,t+1}/\mathsf{st}_{R,t+1}$. *On input a sending or receiving state* $\mathsf{st}_{S,t} \in \mathcal{S}_S$*, resp.* $\mathsf{st}_{R,t} \in \mathcal{S}_R$*, this deterministic algorithm derives the next phase's state* $\mathsf{st}_{S,t+1} \in \mathcal{S}_S$*, resp.* $\mathsf{st}_{R,t+1} \in \mathcal{S}_R$.

  It further employs the (same) deterministic algorithm $\mathsf{KeyDerive}$ *as given for* Init *to derive an updated phase key* $\mathsf{K}_{t+1} \in \mathcal{K}$ *as* $\mathsf{K}_{t+1} \leftarrow \mathsf{KeyDerive}(\mathsf{msk}_{t+1})$.

We call a multi-key channel with a deterministic Send algorithm a *deterministic* multi-key channel.

Shorthand notation. Similar to the previous chapters, we introduce the following shorthand notation. Given a sending state $\mathsf{st}_S \in \mathcal{S}_S$, a phase key $\mathsf{K} \in \mathcal{K}$, an integer $\ell \geq 0$, and a vector of messages $\mathbf{m} = (m_1, \ldots, m_\ell) \in (\{0,1\}^*)^\ell$, let $(\mathsf{st}'_S, \mathbf{c}) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}_S, \mathsf{K}, \mathbf{m})$ be shorthand for the sequential execution $(\mathsf{st}^1_S, c_1) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}^0_S, \mathsf{K}, m_1), \ldots, (\mathsf{st}^\ell_S, c_\ell) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}^{\ell-1}_S, \mathsf{K}, m_\ell)$ with $\mathbf{c} = (c_1, \ldots, c_\ell)$, $\mathsf{st}^0_S = \mathsf{st}_S$, and $\mathsf{st}'_S = \mathsf{st}^\ell_S$. For $\ell = 0$ we define $\mathbf{c}$ to be the empty vector and the final state $\mathsf{st}^\ell_S = \mathsf{st}'_S$ to be the initial state st_S. We use an analogous notation for the Recv algorithm.

Correctness of multi-key channels intuitively guarantees that if at the receiver side the keys are updated after having received all messages sent in the previous phase, then the received messages are equal to those sent in the entire communication.

Definition 12.2 (Correctness of multi-key channels)**.** *Let* $t \in \mathbb{N}$ *and* $(\mathsf{msk}_0, \mathsf{K}_0, \mathsf{st}_{S,0}, \mathsf{st}_{R,0}) \xleftarrow{\$} \mathsf{Init}(1^\lambda)$*. Let* $\mathbf{m}_0, \ldots, \mathbf{m}_t \in (\{0,1\}^*)^*$ *be* $t+1$ *vectors of messages of lengths* $|\mathbf{m}_i| \leq \mathsf{maxmsg}$ *(for* $i \in \{0, \ldots, t\}$*). Let* $\mathbf{c_0}, \ldots, \mathbf{c_t} \in (\{0,1\}^*)^*$ *be the corresponding ciphertext vectors output by* Send *given that* Update *is invoked between each sending of two subsequent message sequences, i.e., such that for* $k = 0, \ldots, t$, $(\mathsf{st}'_{S,k}, \mathbf{c_k}) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}_{S,k}, \mathsf{K}_k, \mathbf{m}_k)$ *and for* $k = 0, \ldots, t-1$, $(\mathsf{msk}_{k+1}, \mathsf{K}_{k+1}, \mathsf{st}_{S,k+1}) \leftarrow \mathsf{Update}(\mathsf{msk}_k, \mathsf{st}'_{S,k})$.

Now let $\mathbf{m}'_0, \ldots, \mathbf{m}'_t \in (\{0,1\}^*)^*$ *be the results of receiving these ciphertexts with likewise interleaved* Update *invocations on the receiver's side, i.e., for* $k = 0, \ldots, t$*, let* $(\mathsf{st}'_{R,k}, \mathbf{m}'_k) \leftarrow \mathsf{Recv}(\mathsf{st}_{R,k}, \mathsf{K}_k, \mathbf{c_k})$ *and for* $k = 0, \ldots, t-1$*, let* $(\mathsf{msk}_{k+1}, \mathsf{K}_{k+1}, \mathsf{st}_{R,k+1}) \leftarrow \mathsf{Update}(\mathsf{msk}_k, \mathsf{st}'_{R,k})$.

We say that a multi-key channel Ch *is* correct *if for any choice of* t*,* $\mathbf{m}_0$*, ...,* $\mathbf{m}_t$*, and all choices of the randomness in the channel algorithms it holds that* $\mathbf{m}_0 = \mathbf{m}'_0$*, ...,* $\mathbf{m}_t = \mathbf{m}'_t$.

12.3 Security of Multi-key Channels

Classically, two security properties are expected from a secure channel. *Confidentiality* aims at protecting the content of transported messages from being read by eavesdroppers or active adversaries on the network. In contrast, *integrity* ensures that messages are received unmodified and in correct order, i.e., without messages being reordered or intermediate messages being dropped. We take up these notions in the context of multi-key channels and extend them to capture two more advanced security aspects arising in this scenario which we denote as *forward security* and *phase-key insulation*.

Forward security, as established also in other settings, is concerned with the effects leaking a channel's master secret key has on prior communication. The notion aims at situations where all key material of a communicating party becomes known to an attacker, e.g., through a break-in into a system or exfiltration of secrets. Following common terminology, we demand that a forward-secure multi-key channel upholds both confidentiality and integrity for messages sent in phases *before* corruption of a master secret key took place, even if one endpoint of the channel is still processing data in these phases when the corruption happens. Naturally, as the deterministic key schedule implies that the current and any future phase's key can be derived from a master secret key, we cannot expect confidentiality or integrity for messages sent from the point of corruption on.

Phase-key insulation in contrast captures the selective leakage of some phases' keys while the master secret key remains uncompromised. Such leakage may be due to cryptanalysis of some of these keys, partial misuse of the key material, or temporary compromise. In particular, it reflects that the master secret key of a channel may be stored in more secure memory (e.g., trusted hardware) while the current phase key potentially resides in lesser-secured memory for performance reasons. From a phase-key–insulated multi-key channel we demand, on a high level, that confidentiality and integrity in a certain phase is not endangered by the leakage of keys in prior or later phases.

12.3.1 Confidentiality

Recall that the established way of modeling confidentiality for channels is by demanding that the encryptions of two (left and right) sequences of messages are indistinguishable [GM84, BKN04], see Section 9.3.1. Formally, an adversary sequentially inputs pairs of messages m_0, m_1 of its choice to a sending oracle $\mathcal{O}_{\mathsf{Send}}$ and is given the encryption c_b of always either the first or the second message depending on an initially fixed, random challenge bit $b \xleftarrow{\$} \{0,1\}$. The adversary's task is to finally determine b. Hence, the corresponding security notion is established under the name of (stateful) indistinguishability under chosen-plaintext attacks (IND-sfCPA). In the stronger setting of chosen-ciphertext attacks (IND-sfCCA), the adversary is additionally given a receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ with the limitation that it may not query it on challenge ciphertexts, defined via a synchronization mechanism (cf. Section 9.3.1).

In the multi-key setting however, the advanced security aspects of forward security and particularly phase-key insulation render it impossible to use a single challenge bit throughout all phases. An adversary that adaptively learns keys for some phases is immediately able to learn whether the left or the right messages were encrypted in these phases. If this would be a fixed choice for all phases, the adversary could also tell which messages were encrypted in all other phases. In our formalization of multi-key confidentiality we hence deploy a separate challenge bit b_i for each phase i, chosen independently at random. This allows us to capture the expected insulation of phases against compromises in other phases and, ultimately, later corruption.

We define confidentiality in a modular notion s-IND-kATK through the security experiment $\mathsf{Expt}^{s\text{-IND-}k\mathsf{ATK}}_{\mathsf{Ch},\mathcal{A}}$ given in Figure 12.2. The experiment is parameterized with the following

parameters s, k, and ATK.

- The parameter s specifies the advanced security aspects captured in the notion and can be either empty ($s = \varepsilon$) or take one of the values ki, fs, or fski. As expected, fs indicates that the notion ensures forward security and ki denotes that the notion demands phase-key insulation; for fski both properties are integrated and the empty string ε indicates a plain notion without forward security or phase-key insulation.[63] Forward security is modeled through allowing the adversary to learn the master secret key at some point through a corruption oracle $\mathcal{O}_{\mathsf{Corrupt}}$. When ensuring phase-key insulation, the adversary is given a reveal oracle $\mathcal{O}_{\mathsf{Reveal}}$ which enables it to selectively learn the keys of some phases.

- Via the parameter k, we capture both single-key (sk) and multi-key (mk) security notions in a single experiment. To model the single-key setting, we simply drop the adversary's capability to proceed to a next phase via an $\mathcal{O}_{\mathsf{Update}}$ oracle, essentially restricting it to a single phase (and hence key).

- Finally, the parameter ATK distinguishes between chosen-plaintext ($\mathsf{ATK} = \mathsf{CPA}$) and chosen-ciphertext ($\mathsf{ATK} = \mathsf{CCA}$) attacks. While the adversary always has access to a left-or-right encryption oracle $\mathcal{O}_{\mathsf{LoR}}$, the receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ is only available for CCA notions.

The adversary finally has to output a phase t and a bit guess b. It wins if the challenge bit used in phase t by the left-or-right oracle $\mathcal{O}_{\mathsf{LoR}}$ is equal to b and the targeted challenge phase t is neither revealed ($t \notin \mathit{Rev}$, where Rev is the set of all revealed phases) nor affected by corruption (i.e., $t < t_{\mathit{corr}}$, where t_{corr} is the corrupted phase, initialized to infinity).

In order to prevent trivial attacks, we have to restrict the output of adversarial queries to the receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ in the setting of chosen-ciphertext attacks. Obviously, if $\mathcal{O}_{\mathsf{Recv}}$ were to output the message decrypted on input the unmodified challenge ciphertext sequence, the challenge bit used in $\mathcal{O}_{\mathsf{LoR}}$ would be immediately distinguishable. Still, as the Recv algorithm is stateful, we must allow the adversary to first make this algorithm proceed to a certain, potentially vulnerable state, before mounting its attack. For this purpose, we (as in the previous chapters) follow the concept of Bellare et al. [BKN04] to suppress the output of the Recv algorithm as long as the adversary's inputs to $\mathcal{O}_{\mathsf{Recv}}$ are *in sync* with the challenge ciphertext sequence output by $\mathcal{O}_{\mathsf{LoR}}$. As soon as synchronization is lost though, $\mathcal{O}_{\mathsf{Recv}}$ returns the output of the receiving algorithm Recv to the adversary.

Defining what it means to be in sync now becomes the crucial task in defining CCA security: we want to make the security notion as strong as possible without allowing trivial attacks. Intuitively, $\mathcal{O}_{\mathsf{Recv}}$ stays in sync (denoted by a flag $\mathsf{sync} = 1$) and decryptions are suppressed as long as the adversary queries ciphertexts to $\mathcal{O}_{\mathsf{Recv}}$ that are obtained in that order from $\mathcal{O}_{\mathsf{LoR}}$ in the same phase. So far, this is essentially a transcription of the stateful encryption definition of CCA security, $\mathsf{IND\text{-}sfCCA}$ (cf. Section 9.3.1), to the multi-key setting with multiple phases. When targeting forward security and phase-key insulation, we however also need to consider how to define synchronization in phases where the adversary knows the key. Obviously, in such phases we cannot demand that a channel can strictly distinguish adversarial encryptions from the honest ciphertext sequence generated in $\mathcal{O}_{\mathsf{LoR}}$ as the adversary may simply replicate the latter's behavior. We accordingly do not consider synchronization to become lost in revealed phases. Still, we demand that a secure channel notices modifications later in uncompromised phases. Moreover, it should even detect truncations at the end of an uncompromised phase if the next phase's key is revealed, latest when the channel recovers from temporary compromise and enters

[63] For legibility, we also drop the leading dash in a notion s-IND-$k\mathsf{ATK}$ if $s = \varepsilon$ is the empty string and simply write IND-$k\mathsf{ATK}$ in this case.

$\underline{\mathsf{Expt}^{s\text{-}\mathsf{IND}\text{-}k\mathsf{ATK}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)}$:
1 $(\mathsf{msk}_0, \mathsf{K}_0, \mathsf{st}_S, \mathsf{st}_R) \xleftarrow{\$} \mathsf{Init}(1^\lambda)$
2 $t_S \leftarrow 0,\ t_R \leftarrow 0$
3 $b_0 \xleftarrow{\$} \{0,1\}$
4 $i_0 \leftarrow 0,\ j_0 \leftarrow 0$
5 $\mathsf{sync} \leftarrow 1$
6 $t_{corr} \leftarrow +\infty$
7 $Rev \leftarrow \emptyset$
8 $(t, b) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}},\ [\mathcal{O}_{\mathsf{Recv}}]_{\mathsf{ATK}=\mathsf{CCA}},\ [\mathcal{O}_{\mathsf{Update}}]_{k=\mathsf{mk}},\ [\mathcal{O}_{\mathsf{Reveal}}]_{s\in\{\mathsf{ki},\mathsf{fski}\}},\ [\mathcal{O}_{\mathsf{Corrupt}}]_{s\in\{\mathsf{fs},\mathsf{fski}\}}}(1^\lambda)$
9 if $t > \max(t_S, t_R)$ then
10 return 0
11 return $((b_t = b) \wedge (t \notin Rev) \wedge (t < t_{corr}))$

$\underline{\mathcal{O}_{\mathsf{LoR}}(m_0, m_1)}$:
12 if $|m_0| \neq |m_1|$ then
13 return $\lightning$
14 $i_{t_S} \leftarrow i_{t_S} + 1$
15 $(\mathsf{st}_S, \mathbf{C}[t_S][i_{t_S}]) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}_S, \mathsf{K}_{t_S}, m_{b_{t_S}})$
16 if $t_R > t_S$ and $t_S \notin Rev$ then
17 $\mathsf{sync} \leftarrow 0$
18 return $\mathbf{C}[t_S][i_{t_S}]$

$\underline{\mathcal{O}_{\mathsf{Recv}}(c)}$:
19 $j_{t_R} \leftarrow j_{t_R} + 1$
20 $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, \mathsf{K}_{t_R}, c)$
21 if $(t_R > t_S$ or $j_{t_R} > i_{t_R}$ or $c \neq \mathbf{C}[t_R][j_{t_R}])$ and $t_R \notin Rev$ then
22 $\mathsf{sync} \leftarrow 0$
23 if $\mathsf{sync} = 0$ then
24 return m
25 else
26 return $\lightning$

$\underline{\mathcal{O}_{\mathsf{Update}}(role)}$:
27 $(\mathsf{msk}_{t_{role}+1}, \mathsf{K}_{t_{role}+1}, \mathsf{st}_{role}) \leftarrow \mathsf{Update}(\mathsf{msk}_{t_{role}}, \mathsf{st}_{role})$
28 if $role = R$ and $t_S \geq t_R$ and $j_{t_R} < i_{t_R}$ and $t_R \notin Rev$ then
29 $\mathsf{sync} \leftarrow 0$
30 $t_{role} \leftarrow t_{role} + 1$
31 $\mathsf{st}^{begin}_{role,t_{role}} \leftarrow \mathsf{st}_{role}$
32 if $role = S$ then
33 $b_{t_S} \xleftarrow{\$} \{0,1\}$

$\underline{\mathcal{O}_{\mathsf{Reveal}}(t, role)}$:
34 if $t > t_{role}$ then
35 return $\lightning$
36 $Rev \leftarrow Rev \cup \{t\}$
37 return $(\mathsf{st}^{begin}_{role,t}, \mathsf{K}_t)$

$\underline{\mathcal{O}_{\mathsf{Corrupt}}(role)}$:
38 if $t_{corr} < +\infty$ then
39 return $(\mathsf{st}^{begin}_{role,t_{corr}}, \mathsf{msk}_{t_{corr}})$
40 $t_{corr} \leftarrow t_{role}$
41 return $(\mathsf{st}^{begin}_{role,t_{role}}, \mathsf{msk}_{t_{role}})$

Figure 12.2: Security experiment for *confidentiality* (s-IND-kATK) of multi-key channels. The brackets $[\mathcal{O}_{\mathsf{X}}]_c$ indicate that the adversary has access to the $\mathcal{O}_{\mathsf{X}}$ oracle only if the condition c is satisfied.

the next, uncompromised phase.[64] We hence, additionally to the regular stateful encryption setting, define synchronization to be lost if the receiver proceeds from an uncompromised phase to the next phase without having received all sent ciphertexts, or if the sender issues a ciphertext in a phase when the receiver already proceeded to the next phase.

In the following we describe the functionality and purpose of the oracles in the multi-key confidentiality experiment in Figure 12.2 in detail.

- The $\mathcal{O}_{\mathsf{LoR}}$ oracle can be queried with a pair of messages (m_0, m_1) of equal length. It responds with the output of Send on message $m_{b_{t_S}}$, where b_{t_S} is the challenge bit for the current sending phase t_S.

  If the receiver already proceeded to a later phase, the sent message cannot be received correctly anymore. As long as the key of the sender's phase is unrevealed, we hence declare

[64] Recall that we consider key updates to be unauthenticated, possibly transmitted out-of-band.

synchronization to be lost (setting $\mathsf{sync} \leftarrow 0$). The restriction to uncompromised phases is to prevent trivial attacks where the adversary leverages the phase key to, e.g., make the receiver process more messages than sent earlier to cover up the mismatch.

- The $\mathcal{O}_{\mathsf{Recv}}$ oracle can only be queried if $\mathsf{ATK} = \mathsf{CCA}$. On input a ciphertext c, $\mathcal{O}_{\mathsf{Recv}}$ computes the corresponding messages obtained under Recv. In case the receiving oracle is ahead in phase, has received more messages than sent, or c deviates from the corresponding sent ciphertext, synchronization is lost (again, to ignore trivial forgeries, as long the receiver's current phase is unrevealed). Finally, if still in sync, $\mathcal{O}_{\mathsf{Recv}}$ suppresses the message output and returns an according flag $\lightning$ to the adversary $\mathcal{A}$. Otherwise it provides $\mathcal{A}$ with the obtained message m.

- The $\mathcal{O}_{\mathsf{Update}}$ oracle is only available if $k = \mathsf{mk}$. Using the oracle, the adversary can separately make both the sender or receiver proceed to the next phase (indicating sender and receiver through $role = S$, resp. $role = R$), updating their master secret, phase key, and state. If the sender side is updated, a fresh challenge bit $b_{t_S} \xleftarrow{\$} \{0,1\}$ for the new phase t_S is chosen at random. The experiment goes out of sync if the receiver side is updated too soon, i.e., without having received all sent ciphertexts, and the receiver's phase is not revealed. Furthermore, the next phase's initial state is stored in $\mathsf{st}^{begin}_{role,t_{role}}$ for answering potential future reveal and corrupt queries.

- The $\mathcal{O}_{\mathsf{Reveal}}$ oracle can be used by the adversary to obtain the key of any phase t (along with this phase's initial sender resp. receiver state) and is accessible if $s \in \{\mathsf{ki}, \mathsf{fski}\}$. Phase t is then added to the set of revealed phases Rev.

- The $\mathcal{O}_{\mathsf{Corrupt}}$ oracle is provided if $s \in \{\mathsf{fs}, \mathsf{fski}\}$. Upon the first call, the adversary obtains for a chosen role $role$ the current phase's master secret key and initial state. This phase is then recorded as the phase of corruption t_{corr} for later comparison. If a corruption has already taken place (i.e., $t_{corr} < +\infty$), the adversary can obtain the other role's initial state in the corrupted phase via a further $\mathcal{O}_{\mathsf{Corrupt}}$ call. For simplicity, we assume the state to be empty in phases not yet entered. Observe that it suffices to consider a single point in time for corruption, as later master keys are deterministically derived from the corrupted one.

We are now ready to state the formal generic definition of s-IND-kATK security for multi-key channels.

Definition 12.3 (s-IND-kATK security)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv}, \mathsf{Update})$ *be a multi-key channel and experiment* $\mathsf{Expt}^{s\text{-IND-}k\mathsf{ATK}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 12.2, parameterized in three dimensions by* s*,* k*, and* ATK *as specified above.*

We say that Ch *provides* indistinguishability under multi-key (resp. single-key) chosen-plaintext (resp. chosen-ciphertext) attacks *(*s-IND-kCPA*, resp.* s-IND-kCCA*, for* $k = \mathsf{mk}$*, resp.* $k = \mathsf{sk}$*), potentially* with forward security *(if* $s \in \{\mathsf{fs}, \mathsf{fski}\}$*)* and/or phase-key insulation *(if* $s \in \{\mathsf{ki}, \mathsf{fski}\}$*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{s\text{-IND-}k\mathsf{ATK}}_{\mathsf{Ch},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{s\text{-IND-}k\mathsf{ATK}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right] - \frac{1}{2}.$$

Our generic confidentiality notion in Definition 12.3 captures as its weakest variant indistinguishability under single-key chosen-plaintext attacks (IND-skCPA) and as its strongest variant indistinguishability under multi-key chosen-ciphertext attacks with forward security and phase-key insulation (fski-IND-mkCCA). We discuss the relations among these notions in more detail in Section 12.3.3.

Remark 12.4. As pointed out earlier, using a single challenge bit across all phases in the given confidentiality experiment is infeasible: an adaptive Reveal query for some phase would in this case also disclose the challenge phase's (same) bit. We hence deploy multiple, independent challenge bits for each phase.

Alternative options would be to employ a single challenge bit in one phase and provide regular (non–LoR) encryption oracles for all other phases, or to have the adversary choose whether to compromise a phase at its beginning. We however deem these approaches not only more complex, but most importantly less adaptive, as they prevent the adversary from retrospectively choosing (non-)challenge phases.

12.3.2 Integrity

Integrity is traditionally defined in two flavors: integrity of plaintexts (INT-PTXT) and integrity of ciphertexts (INT-CTXT), see Chapter 9 for definitions in the stateless and stateful setting [BN00, BSWW13, BKN04]. Integrity of plaintexts intuitively ensures that no adversary is able to make the receiver output a valid message that differs from the previously sent (sequence of) messages. The stronger notion of ciphertext integrity ensures that no adversary can make the receiver output any valid, even recurring message by inputting a forged or modified ciphertext.

Similarly to confidentiality, we define a modular multi-key integrity notion s-INT-kATK, given through the experiment $\mathsf{Expt}^{s\text{-}\mathsf{INT}\text{-}k\mathsf{ATK}}_{\mathsf{Ch},\mathcal{A}}$ in Figure 12.3. Again, the notion is parameterized to integrate forward security and phase-key insulation (via s), the single- and multi-key setting (via k), as well as the two attack targets, $\mathsf{ATK} = \mathsf{PTXT}$ and $\mathsf{ATK} = \mathsf{CTXT}$. An adversary $\mathcal{A}$ in the experiment $\mathsf{Expt}^{s\text{-}\mathsf{INT}\text{-}k\mathsf{ATK}}_{\mathsf{Ch},\mathcal{A}}$ has access to a sending oracle $\mathcal{O}_{\mathsf{Send}}$ (in contrast to confidentiality without left-or-right functionality), one of two receiving oracles $\mathcal{O}^{\mathsf{ATK}}_{\mathsf{Recv}}$ depending on ATK, and—depending on the advanced security properties and key setting captured—oracles $\mathcal{O}_{\mathsf{Update}}$ (without setting a new challenge bit), and $\mathcal{O}_{\mathsf{Reveal}}$ and $\mathcal{O}_{\mathsf{Corrupt}}$, identical to those for confidentiality. In the integrity experiment, the adversary does not provide a particular challenge output, but instead needs to trigger a winning flag win to be set within the experiment run.

Beyond the sending oracle $\mathcal{O}_{\mathsf{Send}}$ only taking and encrypting a single message, the major difference to the confidentiality setting lies in the definition of the $\mathcal{O}^{\mathsf{ATK}}_{\mathsf{Recv}}$ oracle, which in particular comprises the winning condition check. Depending on the attack target, the adversary has access to either the $\mathcal{O}^{\mathsf{PTXT}}_{\mathsf{Recv}}$ or the $\mathcal{O}^{\mathsf{CTXT}}_{\mathsf{Recv}}$ variant of the receiving oracle. Both oracles first of all obtain a ciphertext c and provide the adversary $\mathcal{A}$ with the decrypted message m output by Recv on that ciphertext. Beyond this, they differ in assessing whether $\mathcal{A}$ has succeeded in breaking plaintext, resp. ciphertext, integrity (in which case they set $\mathsf{win} \leftarrow 1$).

- The $\mathcal{O}^{\mathsf{PTXT}}_{\mathsf{Recv}}$ oracle declares the adversary successful if the received message m differs from the corresponding sent message in this phase and position, given that the current receiving phase is neither revealed nor corrupted.

- The $\mathcal{O}^{\mathsf{CTXT}}_{\mathsf{Recv}}$ in contrast requires for winning that, on input an out-of-sync ciphertext in a phase neither revealed nor corrupted, Recv outputs a valid (non-error) message $m \notin \mathcal{E}$.

  In the same way as for confidentiality, synchronization is considered to be lost in an $\mathcal{O}^{\mathsf{CTXT}}_{\mathsf{Recv}}$ oracle call if the receiving oracle, in a non-revealed phase, is ahead of the sending oracle in phase or message count, or if c deviates from the corresponding sent message. Furthermore, synchronization may be lost by non-aligned key updates on both sides of the channel, captured in $\mathcal{O}_{\mathsf{Send}}$ and $\mathcal{O}_{\mathsf{Update}}$ as in the confidentiality experiment (cf. Figure 12.2).

The generic notion of s-INT-kATK security for multi-key channels is then defined as follows.

$\underline{\mathsf{Expt}^{s\text{-}\mathsf{INT}\text{-}k\mathsf{ATK}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)}$:
1 $(\mathsf{msk}_0, \mathsf{K}_0, \mathsf{st}_S, \mathsf{st}_R) \xleftarrow{\$} \mathsf{Init}(1^\lambda)$
2 $t_S \leftarrow 0$, $t_R \leftarrow 0$, $i_0 \leftarrow 0$, $j_0 \leftarrow 0$
3 $\mathsf{sync} \leftarrow 1$, $\mathsf{win} \leftarrow 0$
4 $t_{corr} \leftarrow +\infty$, $Rev \leftarrow \emptyset$
5 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}},\ \mathcal{O}^{\mathsf{ATK}}_{\mathsf{Recv}},\ [\mathcal{O}_{\mathsf{Update}}]_{k=\mathsf{mk}},\ [\mathcal{O}_{\mathsf{Reveal}}]_{s\in\{\mathsf{ki},\mathsf{fski}\}},\ [\mathcal{O}_{\mathsf{Corrupt}}]_{s\in\{\mathsf{fs},\mathsf{fski}\}}}(1^\lambda)$
6 return win

$\underline{\mathcal{O}_{\mathsf{Send}}(m)}$:
7 $i_{t_S} \leftarrow i_{t_S} + 1$
8 $(\mathsf{st}_S, \mathbf{C}[t_S][i_{t_S}]) \xleftarrow{\$} \mathsf{Send}(\mathsf{st}_S, \mathsf{K}_{t_S}, m)$
9 $\mathbf{M}[t_S][i_{t_S}] \leftarrow m$
10 if $t_R > t_S$ and $t_S \notin Rev$ then
11 $\quad \mathsf{sync} \leftarrow 0$
12 return $\mathbf{C}[t_S][i_{t_S}]$

$\underline{\mathcal{O}^{\mathsf{PTXT}}_{\mathsf{Recv}}(c)}$:
13 $j_{t_R} \leftarrow j_{t_R} + 1$
14 $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, \mathsf{K}_{t_R}, c)$
15 if $m \neq \mathbf{M}[t_R][j_{t_R}]$ and $m \notin \mathcal{E}$ and $t_R \notin Rev$
and $t_R < t_{corr}$ then
16 $\quad \mathsf{win} \leftarrow 1$
17 return m

$\underline{\mathcal{O}^{\mathsf{CTXT}}_{\mathsf{Recv}}(c)}$:
18 $j_{t_R} \leftarrow j_{t_R} + 1$
19 $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, \mathsf{K}_{t_R}, c)$
20 if $(t_R > t_S$ or $j_{t_R} > i_{t_R}$ or $c \neq \mathbf{C}[t_R][j_{t_R}])$
and $t_R \notin Rev$ then
21 $\quad \mathsf{sync} \leftarrow 0$
22 if $\mathsf{sync} = 0$ and $m \notin \mathcal{E}$ and $t_R \notin Rev$
and $t_R < t_{corr}$ then
23 $\quad \mathsf{win} \leftarrow 1$
24 return m

$\underline{\mathcal{O}_{\mathsf{Update}}(role)}$:
25 $(\mathsf{msk}_{t_{role}+1}, \mathsf{K}_{t_{role}+1}, \mathsf{st}_{role}) \leftarrow \mathsf{Update}(\mathsf{msk}_{t_{role}}, \mathsf{st}_{role})$
26 if $role = R$ and $t_S \geq t_R$ and $j_{t_R} < i_{t_R}$
and $t_R \notin Rev$ then
27 $\quad \mathsf{sync} \leftarrow 0$
28 $t_{role} \leftarrow t_{role} + 1$
29 $\mathsf{st}^{begin}_{role,t_{role}} \leftarrow \mathsf{st}_{role}$

$\underline{\mathcal{O}_{\mathsf{Reveal}}(t, role)}$:
30 if $t > t_{role}$ then
31 $\quad$ return $\lightning$
32 $Rev \leftarrow Rev \cup \{t\}$
33 return $(\mathsf{st}^{begin}_{role,t}, \mathsf{K}_t)$

$\underline{\mathcal{O}_{\mathsf{Corrupt}}(role)}$:
34 if $t_{corr} < +\infty$ then
35 $\quad$ return $(\mathsf{st}^{begin}_{role,t_{corr}}, \mathsf{msk}_{t_{corr}})$
36 $t_{corr} \leftarrow t_{role}$
37 return $(\mathsf{st}^{begin}_{role,t_{role}}, \mathsf{msk}_{t_{role}})$

Figure 12.3: Security experiment for *integrity* (s-INT-kATK) of multi-key channels. The brackets $[\mathcal{O}_{\mathsf{X}}]_c$ indicate that the adversary has access to the $\mathcal{O}_{\mathsf{X}}$ oracle only if the condition c is satisfied; the $\mathcal{O}_{\mathsf{Recv}}$ differs depending on the parameter $\mathsf{ATK} \in \{\mathsf{PTXT}, \mathsf{CTXT}\}$.

Definition 12.5 (s-INT-kATK security)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv}, \mathsf{Update})$ *be a multi-key channel and experiment* $\mathsf{Expt}^{s\text{-}\mathsf{INT}\text{-}k\mathsf{ATK}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 12.3, parameterized in three dimensions by* s*,* k*, and* ATK *as specified above.*

We say that Ch *provides* multi-key (resp. single-key) integrity of plaintexts (resp. ciphertexts) *(*s-INT-kPTXT*, resp.* s-INT-kCTXT*, for* $k = \mathsf{mk}$*, resp.* $k = \mathsf{sk}$*), potentially* with forward security *(if* $s \in \{\mathsf{fs}, \mathsf{fski}\}$*)* and/or phase-key insulation *(if* $s \in \{\mathsf{ki}, \mathsf{fski}\}$*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{s\text{-}\mathsf{INT}\text{-}k\mathsf{ATK}}_{\mathsf{Ch},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{s\text{-}\mathsf{INT}\text{-}k\mathsf{ATK}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right].$$

12.3.3 Relations Between Multi- and Single-key Notions

The modularity of our notions for multi-key confidentiality and integrity, parameterized by forward security and phase-key insulation, leads to a set of notions of varying strength. In the

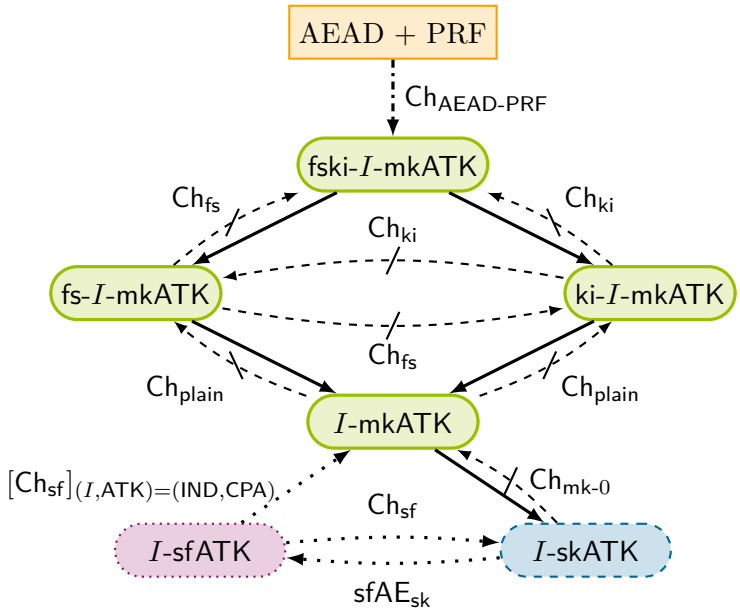


Figure 12.4: Illustration of the relations between different flavors of confidentiality and integrity in our multi-key and single-key settings as well as for stateful authenticated encryption [BKN04] (cf. Section 9.3). The variables I and ATK are placeholders for confidentiality notions ($I = \mathsf{IND}$ with $\mathsf{ATK} = \mathsf{CPA}/\mathsf{CCA}$) and integrity notions ($I = \mathsf{INT}$ with $\mathsf{ATK} = \mathsf{PTXT}/\mathsf{CTXT}$).
Rounded rectangles indicate multi-key (solid-line, green), single-key (dashed-line, blue), or stateful-AE notions (dotted-line, purple); regular (orange) rectangles indicate building blocks. Solid arrows indicate trivial implications (through omission of oracles in the respective experiment). Dashed, stroke-out arrows indicate separations and dotted arrows generic transforms we establish, both provided in Section 12.3.3. The dash-dotted arrow indicates the generic $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ construction we provide in Section 12.4. Labels on arrows refer to the respective construction providing the implication, separation, or transform. For a construction X in brackets $[\mathsf{X}]_c$, the relation only holds between notions for which the condition c is satisfied.

following, we establish that forward security and phase-key insulation are orthogonal properties (expectedly both adding to the strength of a security notion) and only take effect in the multi-key setting. Furthermore, we show that the single-key security notions of our framework are essentially equivalent to the respective established stateful encryption notions: we give generic, purely syntactical transforms to translate secure single-key schemes between the two realms. Figure 12.4 illustrates the relations we establish.

Trivial implications

First of all, let us observe the trivial implications between the security notions of our framework, indicated by solid arrows in Figure 12.4. Those implications arise by restricting the access to one (or multiple) oracles in the security experiments: a notion with access to a certain oracle immediately implies an otherwise identical notion without this oracle access. For instance, a fski-IND-mkCPA-secure channel is also ki-IND-mkCPA-secure, since if no adversary can distinguish left-or-right ciphertexts when being able to corrupt the master secret key, then doing so does not become easier when corruption is not possible.

We furthermore observe that the advanced properties of forward security and phase-key insulation are only reasonable to consider in the multi-key setting ($k = \mathsf{mk}$), for both confidentiality and integrity. Indeed, for the single-key setting ($k = \mathsf{sk}$), the plain, fs, ki, and fski flavors of each notion collapse to being equivalent. For this, observe that an adversary in the single-key

setting, lacking access to the $\mathcal{O}_{\mathsf{Update}}$ oracle, is restricted to the initial phase $t_S = t_R = 0$. At the same time, in order to win in this phase (by outputting a confidentiality guess, resp. breaking integrity), it must neither reveal the single phase key used nor corrupt either of the parties. Hence, the adversary effectively cannot make use of the $\mathcal{O}_{\mathsf{Reveal}}$ and $\mathcal{O}_{\mathsf{Corrupt}}$ queries, rendering both non-effective. Consequently, we can focus on the plain version of our single-key security notions only, as reflected in Figure 12.4.

Separations

We discuss the separations between notions possibly providing forward security and phase-key insulation starting from a multi-key channel that provides both properties at the example of indistinguishability under chosen-plaintext attacks. The cases of integrity and indistinguishability under chosen-ciphertext attacks are analogous. More precisely, let $\mathsf{Ch}_{\mathsf{fski}} := (\mathsf{Init}_{\mathsf{fski}}, \mathsf{Send}_{\mathsf{fski}}, \mathsf{Recv}_{\mathsf{fski}}, \mathsf{Update}_{\mathsf{fski}})$ be a multi-key channel which provides fski-IND-mkCPA security. Recall that master secret and phase keys are computed using two deterministic sub-algorithms $\mathsf{MasterKeyUp}_{\mathsf{fski}}$ and $\mathsf{KeyDerive}_{\mathsf{fski}}$, respectively.

First, we construct a new channel $\mathsf{Ch}_{\mathsf{fs}}$ which differs from $\mathsf{Ch}_{\mathsf{fski}}$ only in its key derivation algorithm KeyDerive which we replace by the identity function, i.e., we define $\mathsf{KeyDerive}_{\mathsf{fs}}(\mathsf{msk}_i) := \mathsf{msk}_i$ for all phases $i \in \mathbb{N}$. As MasterKeyUp remains unmodified, master or phase keys can not be used to compute previous phases' keys, so $\mathsf{Ch}_{\mathsf{fs}}$ inherits the forward security of $\mathsf{Ch}_{\mathsf{fski}}$. However, observe that a revealed phase key (equal to the master secret key $\mathsf{K}_i = \mathsf{msk}_i$) can now be iteratively used to compute the next master secret keys $\mathsf{msk}_{i+1} = \mathsf{MasterKeyUp}_{\mathsf{fs}}(\mathsf{msk}_i)$ and therefore also the next phase keys $\mathsf{K}_{i+1} = \mathsf{KeyDerive}_{\mathsf{fs}}(\mathsf{msk}_{i+1})$. Thus, $\mathsf{Ch}_{\mathsf{fs}}$ has dependent phase keys and hence only provides fs-IND-mkCPA security, but not fski-IND-mkCPA security, separating the two notions.

Next we build a channel $\mathsf{Ch}_{\mathsf{ki}}$ from $\mathsf{Ch}_{\mathsf{fski}}$ which has a master secret key space $\mathcal{MSK}_{\mathsf{ki}} = \mathcal{MSK}^*_{\mathsf{fski}}$ and updates its master secret keys using a function $\mathsf{MasterKeyUp}_{\mathsf{ki}}(\mathbf{msk}_i) := (\mathbf{msk}_i, \mathsf{MasterKeyUp}_{\mathsf{fski}}(\mathbf{msk}_i[i]))$, where $\mathbf{msk}_0 = (\mathsf{MasterKeyGen}_{\mathsf{fski}}(1^\lambda))$. In other words, $\mathsf{Ch}_{\mathsf{ki}}$ keeps a copy of all master secret keys generated so far in the current master secret key, and uses the last entry to derive the next master secret key. The phase keys are then derived from the last master secret key entry, i.e., we define $\mathsf{KeyDerive}_{\mathsf{ki}}(\mathbf{msk}_i) := \mathsf{KeyDerive}_{\mathsf{fski}}(\mathbf{msk}_i[i])$. Observe that $\mathsf{Ch}_{\mathsf{ki}}$ provides the phase-key insulation of $\mathsf{Ch}_{\mathsf{fski}}$: the phase keys are computed and used as before and hence in particular cannot be leveraged to gain information about the now accumulated master secrets, as otherwise phase-key insulation would be broken in $\mathsf{Ch}_{\mathsf{fski}}$ already. Forward security however is lost. On corruption in any phase, all previous master secret keys are leaked, allowing an adversary to derive any previous phase key. Therefore $\mathsf{Ch}_{\mathsf{ki}}$ only provides ki-IND-mkCPA security, but not fski-IND-mkCPA security.

Combining the two modifications above leads to a channel $\mathsf{Ch}_{\mathsf{plain}}$ which only satisfies plain IND-mkCPA security, but provides neither phase-key insulation nor forward security. This hence separates IND-mkCPA from both ki-IND-mkCPA and fs-IND-mkCPA.

Finally, we consider the separation between the single-key notions and their corresponding multi-key notions, both without forward security and phase-key insulation. Again, we only discuss the notions IND-skCPA and IND-mkCPA as an example; the other cases follow identically. We build from an IND-skCPA secure single-key channel $\mathsf{Ch}_{\mathsf{sk}}$ a multi-key channel $\mathsf{Ch}_{\mathsf{mk\text{-}0}}$ which uses the single-key channel's key for the initial phase both as master secret and phase key. As the master secret key for the second and all following phases it then uses the zero-string, i.e., $\mathsf{MasterKeyUp}_{\mathsf{mk}}(\mathsf{msk}_i) := 0^\lambda$. Clearly the security is not preserved by $\mathsf{Ch}_{\mathsf{mk\text{-}0}}$ in any phase other than the initial one, in which it behaves exactly like $\mathsf{Ch}_{\mathsf{sk}}$. Hence, $\mathsf{Ch}_{\mathsf{mk\text{-}0}}$ is IND-skCPA-secure, but not IND-mkCPA-secure.

Generic Transforms Between Stateful Authenticated Encryption and Multi-key Channels

To complete the picture, we finally study the relations between the traditional notion of stateful authenticated encryption (see Section 9.3 for their syntax and security definition) and our notion of multi-key channels.

Clearly, stateful encryption does not aim at achieving the advanced security properties we consider in this work, forward security and phase-key insulation. In our comparison, we hence focus on the plain confidentiality and integrity notions, i.e., IND-kATK and INT-kATK (for both $k \in \{\mathsf{mk}, \mathsf{sk}\}$ and variants $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA}\}$, resp. $\mathsf{ATK} \in \{\mathsf{PTXT}, \mathsf{CTXT}\}$) in our framework as well as the stateful-encryption notions IND-sfCPA, resp. IND-sfCCA, and INT-sfPTXT, resp. INT-sfCTXT (cf. Section 9.3.1).

Indeed, our single-key security notions which allow an adversary to access a multi-key channel only in its initial phase are equivalent in strength to the corresponding stateful-encryption notions, beyond syntactical differences. For this, the following natural and generic transforms for constructing a multi-key channel $\mathsf{Ch}_{\mathsf{sf}}$ from any stateful encryption scheme sfAE and, conversely, a stateful encryption scheme $\mathsf{sfAE}_{\mathsf{sk}}$ from any multi-key channel with single-entry error space $\mathcal{E} = \{\bot\}$.

- $\mathsf{Ch}_{\mathsf{sf}}(\mathsf{Init}_{\mathsf{sf}}, \mathsf{Send}_{\mathsf{sf}}, \mathsf{Recv}_{\mathsf{sf}}, \mathsf{Update}_{\mathsf{sf}})$.
  For initialization, derive $(K, \mathsf{st}_S, \mathsf{st}_R) \xleftarrow{\$} \mathsf{Init}(1^\lambda)$ and set $\mathsf{msk}_0 = \mathsf{K}_0 = K$, $\mathsf{st}_{S,0} = \mathsf{st}_S$, and $\mathsf{st}_{R,0} = \mathsf{st}_R$. For sending and receiving, use Enc and Dec as direct replacements. Finally, the Update algorithm does nothing; i.e., StateUp, MasterKeyUp, and KeyDerive are defined to be the identity function.

- $\mathsf{sfAE}_{\mathsf{sk}}(\mathsf{Init}_{\mathsf{sk}}, \mathsf{Enc}_{\mathsf{sk}}, \mathsf{Dec}_{\mathsf{sk}})$.
  For key generation, derive $(\mathsf{msk}_0, \mathsf{K}_0, \mathsf{st}_{S,0}, \mathsf{st}_{R,0}) \xleftarrow{\$} \mathsf{Init}(1^\lambda)$ and set $K = \mathsf{msk}_0$, $\mathsf{st}_S = \mathsf{st}_{S,0}$, and $\mathsf{st}_R = \mathsf{st}_{R,0}$. Encryption and decryption is directly replaced by Send resp. Recv.

Careful inspection of the single-key ($k = \mathsf{sk}$) notions in our framework and those defined for stateful authenticated encryption (cf. Section 9.3.1) readily establishes that satisfaction of each two corresponding notions (i.e., IND-sfATK and IND-skATK, where $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA}\}$, resp. INT-sfATK and INT-skATK, where $\mathsf{ATK} \in \{\mathsf{PTXT}, \mathsf{CTXT}\}$) is preserved by the generic transforms given above. That is, if the underlying stateful encryption scheme sfAE achieves, e.g., IND-sfCCA security then the transformed multi-key channel $\mathsf{Ch}_{\mathsf{sf}}$ satisfies the corresponding IND-skCCA notion, and vice versa.

Finally, and perhaps surprisingly at first glance, our generic transform $\mathsf{Ch}_{\mathsf{sf}}$ of a stateful encryption scheme into a multi-key channel also achieves (plain) multi-key IND-mkCPA security if the stateful encryption scheme satisfies IND-sfCPA security. The reason for this is that the degenerate Update algorithm does not alter the key which hence also makes the $\mathcal{O}_{\mathsf{Send}}$ oracle not alter its behavior across different phases, thus corresponding directly to the IND-sfCPA setting. In contrast, the message vector $\mathbf{M}$, resp. ciphertext vector $\mathbf{C}$, in the $\mathcal{O}_{\mathsf{Recv}}$ oracle can be easily set out-of-sync by invoking Update at different positions in the ciphertext sequence on the sender and receiver side. As a result, an adversary can make challenge ciphertexts to be considered as valid forgery in a "different" phase (in the multi-key integrity game) or force challenge messages to be output by $\mathcal{O}_{\mathsf{Recv}}$ (in the IND-mkCCA game). Hence, $\mathsf{Ch}_{\mathsf{sf}}$ achieves neither IND-mkCCA nor INT-mkPTXT or INT-mkCTXT security.

12.3.4 Generic Composition

We round up the discussion of our framework of multi-key security notions by lifting the classical composition theorem by Bellare and Namprempre [BN00] for symmetric encryption, namely that

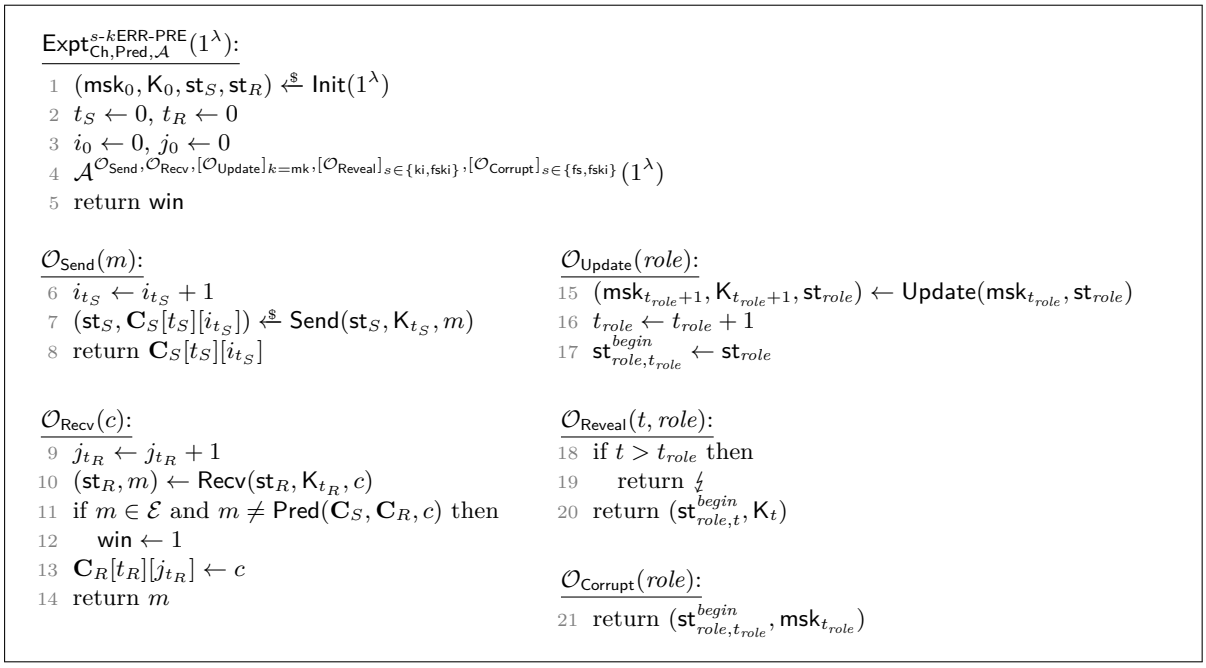


Figure 12.5: Security experiment for *error predictability* (s-kERR-PRE) of multi-key channels. The brackets $[\mathcal{O}_{\mathsf{X}}]_c$ indicate that the adversary has access to the $\mathcal{O}_{\mathsf{X}}$ oracle only if the condition c is satisfied.

IND-CPA and INT-CTXT security imply IND-CCA security, to the setting of multi-key channels. As noted by Boldyreva et al. [BDPS14], this result is not directly applicable in settings where the decryption algorithm may output multiple, distinguishable errors, an observation that also applies to our setting. Boldyreva et al. re-establish composition in the multiple-error setting by requiring that with overwhelming probability an adversary is only able to produce a single error (a notion they call *error invariance*). Here, we instead make use of the more versatile approach introduced earlier in Section 10.3.3 as *error predictability* in the context of our model for stream-based channels (cf. Definition 10.9 on page 161). To recap, error predictability roughly requires that there exists an efficient *predictor* algorithm Pred that, given the ciphertexts sent and received so far, can with overwhelming probability predict the error message caused by receiving a certain next ciphertext (if that ciphertext produces at all an error).

We translate the notion of error predictability to the multi-key setting, parameterized as s-kERR-PRE with forward security and phase-key insulation, and in a single- and multi-key variant. This enables us to show the following composition result: for any advanced security property $s \in \{\varepsilon, \mathsf{fs}, \mathsf{ki}, \mathsf{fski}\}$ and key setting $k \in \{\mathsf{sk}, \mathsf{mk}\}$, if a multi-key channel provides the according notion of ciphertext integrity (s-INT-kCTXT), chosen-plaintext confidentiality (s-IND-kCPA), and error predictability (s-kERR-PRE), then it also provides chosen-ciphertext confidentiality (s-IND-kCCA).

We formalize the parameterized multi-key version of error predictability, s-kERR-PRE, in Definition 12.6 below through the experiment $\mathsf{Expt}^{s\text{-}k\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathcal{A}}$ in Figure 12.5. An adversary wins in this experiment if it can ever cause the Recv algorithm to output an error message that differs from the output of the predictor algorithm. Meanwhile, when forward security or phase-key insulation is demanded, the adversary is also allowed to corrupt the master secret key, resp. to reveal phase keys.

Definition 12.6 (Multi-key error predictability (s-kERR-PRE))**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv}, \mathsf{Update})$ *be a multi-key channel with error space* $\mathcal{E}$*, advanced security aspects* $s \in \{\varepsilon, \mathsf{fs}, \mathsf{ki}, \mathsf{fski}\}$

and key setting $k \in \{\mathsf{sk}, \mathsf{mk}\}$*, and let* Pred *be an efficient probabilistic algorithm. We say that* Ch *provides* (multi-key) error predictability (s-kERR-PRE) with respect to Pred *if for every PPT adversary* $\mathcal{A}$ *playing in the experiment* s-kERR-PRE *defined in Figure 12.5 against channel* Ch*, the following advantage function is negligible:*

$$\mathsf{Adv}^{s\text{-}k\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{s\text{-}k\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{A}}(1^\lambda) = 1\right].$$

We are now ready to state our generic composition theorem for the setting of multi-key channels.

Theorem 12.7 (s-INT-kCTXT$\wedge s$-IND-kCPA$\wedge s$-kERR-PRE $\implies$ s-IND-kCCA)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv}, \mathsf{Update})$ *be a (correct) multi-key channel. If* Ch *provides integrity of ciphertexts (*s-INT-kCTXT*), indistinguishability under chosen-plaintext attacks (*s-IND-kCPA*), and error predictability with respect to some predictor* Pred *(*s-kERR-PRE*) with advanced security aspects* $s \in \{\varepsilon, \mathsf{fs}, \mathsf{ki}, \mathsf{fski}\}$ *for a key setting* $k \in \{\mathsf{sk}, \mathsf{mk}\}$*, then it also provides indistinguishability under chosen-ciphertext attacks for* s *and* k *(*s-IND-kCCA*). Formally, for every efficient* s-IND-kCCA *adversary* $\mathcal{A}$ *there exist an efficient* s-INT-kCTXT *adversary* $\mathcal{B}$*,* s-kERR-PRE *adversary* $\mathcal{C}$*, and* s-IND-kCPA *adversary* $\mathcal{D}$ *such that*

$$\mathsf{Adv}^{s\text{-}\mathsf{IND}\text{-}k\mathsf{CCA}}_{\mathsf{Ch},\mathcal{A}} \leq \mathsf{Adv}^{s\text{-}\mathsf{INT}\text{-}k\mathsf{CTXT}}_{\mathsf{Ch},\mathcal{B}} + \mathsf{Adv}^{s\text{-}k\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{C}} + \mathsf{Adv}^{s\text{-}\mathsf{IND}\text{-}k\mathsf{CPA}}_{\mathsf{Ch},\mathcal{D}}.$$

Proof. By means of intermediate games $\mathsf{E}^0_\mathcal{A}$, $\mathsf{E}^1_\mathcal{A}$, and $\mathsf{E}^2_\mathcal{A}$ we make a transition from the s-IND-kCCA experiment to the s-IND-kCPA experiment in three steps, while bounding the probability differences between each two games with the advantage of a specific adversary.

Let $\mathsf{E}^0_\mathcal{A}$ be the experiment s-IND-kCCA defined in Figure 12.2 against adversary $\mathcal{A}$. Let bad_I be the event that $\mathcal{O}_{\mathsf{Recv}}$ on input a ciphertext outputs a valid message $m \notin \mathcal{E}$ while the receiving phase is neither revealed nor affected by corruption, i.e., $t_R \notin Rev$ and $t_R < t_{corr}$. We define a new experiment $\mathsf{E}^1_\mathcal{A}$ which differs from $\mathsf{E}^0_\mathcal{A}$ in that, within $\mathcal{O}_{\mathsf{Recv}}$, it checks for the bad event before Line 24 and, if bad_I is triggered, replaces m with the output of $\mathsf{Pred}(\mathbf{C}_S, \mathbf{C}_R, c)$ where $\mathbf{C}_S = \mathbf{C}$ and $\mathbf{C}_R$ are the vectors of messages sent, resp. received, prior to the oracle call. By definition, $\mathsf{E}^1_\mathcal{A}$ and $\mathsf{E}^0_\mathcal{A}$ behave equally from $\mathcal{A}$'s perspective, unless bad_I occurs. Using, e.g., $\Pr[\mathsf{E}^0_\mathcal{A}]$ as a shorthand notation for $\Pr[\mathsf{E}^0_\mathcal{A}(1^\lambda) = 1]$, we have:

$$\mathsf{Adv}^{s\text{-}\mathsf{IND}\text{-}k\mathsf{CCA}}_{\mathsf{Ch},\mathcal{A}} = \Pr[\mathsf{E}^0_\mathcal{A}] - \tfrac{1}{2} = \Pr[\mathsf{E}^0_\mathcal{A}] - \Pr[\mathsf{E}^1_\mathcal{A}] + \Pr[\mathsf{E}^1_\mathcal{A}] - \tfrac{1}{2} \leq \Pr[bad_I] + \Pr[\mathsf{E}^1_\mathcal{A}] - \tfrac{1}{2}.$$

We next show how to build from any adversary $\mathcal{A}$ that triggers bad_I an adversary $\mathcal{B}$ that breaks the s-INT-kCTXT security of Ch. Adversary $\mathcal{B}$ keeps an index i initialized to 0 and picks a bit b_0 uniformly at random. It then simulates the s-IND-kCCA experiment for $\mathcal{A}$, answering its queries as follows. If $\mathcal{A}$ queries equal-length messages (m_0, m_1) to $\mathcal{O}_{\mathsf{LoR}}$ then $\mathcal{B}$ queries m_{b_i} to its oracle $\mathcal{O}_{\mathsf{Send}}$ and forwards the answer to $\mathcal{A}$. Similarly $\mathcal{B}$ forwards every receiving query c to its oracle $\mathcal{O}^{\mathsf{CTXT}}_{\mathsf{Recv}}$ and obtains a response m. Depending on the sync flag, $\mathcal{B}$ either forwards m to $\mathcal{A}$ if $\mathsf{sync} = 0$ or responds with $\lightning$ if $\mathsf{sync} = 1$. If $\mathcal{A}$ queries S to $\mathcal{O}_{\mathsf{Update}}$, the adversary $\mathcal{B}$ invokes its own $\mathcal{O}_{\mathsf{Update}}$ oracle on S and additionally increases i by one and chooses a bit b_i uniformly at random. Finally, if $\mathcal{A}$ queries R to $\mathcal{O}_{\mathsf{Update}}$, or queries the oracles $\mathcal{O}_{\mathsf{Reveal}}$ or $\mathcal{O}_{\mathsf{Corrupt}}$, then $\mathcal{B}$ simply relays the queries to its own corresponding oracles and forwards their answer to $\mathcal{A}$. When $\mathcal{A}$ halts, so does $\mathcal{B}$.

Observe that, by definition of the experiments, the sync flag in the simulated s-IND-kCCA experiment for $\mathcal{A}$ and in $\mathcal{B}$'s s-INT-kCTXT experiment coincide. Moreover, if the event bad_I in Line 24 is triggered, i.e., if $\mathsf{sync} = 0$ and $m \notin \mathcal{E}$ in an uncompromised phase, then the winning flag is set in the s-INT-kCTXT for $\mathcal{B}$. Hence, the probability of bad_I being triggered is upper bounded by $\mathcal{B}$'s advantage:

$$\mathsf{Adv}^{s\text{-}\mathsf{INT}\text{-}k\mathsf{CTXT}}_{\mathsf{Ch},\mathcal{B}} \geq \Pr[bad_I].$$

So far we can bound the advantage of $\mathcal{A}$ in the s-IND-kCCA experiment as follows:

$$\mathsf{Adv}^{s\text{-IND-}k\mathsf{CCA}}_{\mathsf{Ch},\mathcal{A}} \leq \Pr[bad_I] + \Pr[\mathsf{E}^1_{\mathcal{A}}] - \tfrac{1}{2} \leq \mathsf{Adv}^{s\text{-INT-}k\mathsf{CTXT}}_{\mathsf{Ch},\mathcal{B}} + \Pr[\mathsf{E}^1_{\mathcal{A}}] - \tfrac{1}{2}.$$

Observe that in game $\mathsf{E}^1_{\mathcal{A}}$, the adversary in case of the bad_I event only obtains the error predictoroutput's output, but no actual messages anymore. We now consider a game $\mathsf{E}^2_{\mathcal{A}}$ by modifying $\mathsf{E}^1_{\mathcal{A}}$ as follows. If the receiving oracle of $\mathsf{E}^2_{\mathcal{A}}$ is in a non-compromised phase, it always uses a predictor Pred instead of Recv to produce the outputs (i.e., also if $m \in \mathcal{E}$). More precisely, we modify $\mathcal{O}_{\mathsf{Recv}}$ by replacing the check for bad_I before Line 24 with a check only for $t_R \notin Rev \wedge t_R \geq t_{corr}$, again followed by a line $m \leftarrow \mathsf{Pred}(\mathbf{C}_S, \mathbf{C}_R, c)$. Let bad_E be the event that the output of Pred differs from an error output of Recv in game $\mathsf{E}^1_{\mathcal{A}}$ in such a non-compromised phase. Then $\mathsf{E}^1_{\mathcal{A}}$ and $\mathsf{E}^2_{\mathcal{A}}$ behave equally as long as bad_E does not occur. Hence we obtain a new bound $|\Pr[\mathsf{E}^1_{\mathcal{A}}] - \Pr[\mathsf{E}^2_{\mathcal{A}}]| \leq \Pr[bad_E]$.

We now show how to build from an adversary $\mathcal{A}$ that triggers bad_E an adversary $\mathcal{C}$ that breaks the s-kERR-PRE property of Ch (wrt. error predictor Pred). Adversary $\mathcal{C}$ keeps an index i initialized to 0 and picks a bit b_0 uniformly at random. It then simulates the game $\mathsf{E}^2_{\mathcal{A}}$ for $\mathcal{A}$ by answering $\mathcal{A}$'s queries using its oracles, analogous to the above adversary $\mathcal{B}$. When $\mathcal{A}$ triggers bad_E in $\mathsf{E}^1_{\mathcal{A}}$, by definition of bad_E it will be due to a deviation of an error output by Recv from the output of the Pred algorithm, thus leading to $\mathcal{C}$ winning in the s-kERR-PRE experiment. Hence we obtain $\mathsf{Adv}^{s\text{-}k\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathcal{C}} \geq \Pr[bad_E]$, which allows us to bound the advantage of $\mathcal{A}$ as follows:

$$\Pr[\mathsf{E}^1_{\mathcal{A}}] = \Pr[\mathsf{E}^1_{\mathcal{A}}] - \Pr[\mathsf{E}^2_{\mathcal{A}}] + \Pr[\mathsf{E}^2_{\mathcal{A}}] \leq \mathsf{Adv}^{s\text{-}k\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{C}} + \Pr[\mathsf{E}^2_{\mathcal{A}}].$$

In the last step we show that with the events bad_I and bad_E being excluded in $\mathsf{E}^2_{\mathcal{A}}$, an adversary $\mathcal{D}$ as defined in Figure 12.6 against the game s-IND-kCPA can simulate the game $\mathsf{E}^2_{\mathcal{A}}$ for $\mathcal{A}$ by answering queries to $\mathcal{O}_{\mathsf{Recv}}$ on its own. To this end, it invokes the predictor Pred whenever the receiving phase is uncompromised and returns its output. Otherwise, for a revealed or corrupted phase, it uses the genuine phase key to compute the output of Recv itself. Observe that invocations of the $\mathcal{O}_{\mathsf{Reveal}}$ or $\mathcal{O}_{\mathsf{Corrupt}}$ query that led to a phase being compromised are relayed through $\mathcal{D}$. Hence, $\mathcal{D}$ in particular knows the phase key of revealed phases and can compute those following a corruption using the obtained compromised master secret key to derive the according phase key via invoking MasterKeyUp and KeyDerive. Furthermore, these queries yield the initial state of compromised phases, allowing $\mathcal{D}$ to proceed the Recv algorithm to any position in the received ciphertext sequence in that phase. All other queries of $\mathcal{A}$ to $\mathcal{O}_{\mathsf{LoR}}$, $\mathcal{O}_{\mathsf{Update}}$, $\mathcal{O}_{\mathsf{Reveal}}$, and $\mathcal{O}_{\mathsf{Corrupt}}$ are relayed by $\mathcal{D}$ to its corresponding oracles in the s-IND-kCPA experiment. When $\mathcal{A}$ stops and outputs a guess (t, b), $\mathcal{D}$ stops outputting the same guess.

We observe that $\mathcal{D}$ provides a correct simulation of $\mathsf{E}^2_{\mathcal{A}}$ for $\mathcal{A}$. Moreover, a valid guess of $\mathcal{A}$ also makes $\mathcal{D}$ win in the s-IND-kCPA experiment. Therefore we obtain the following bound for $\mathcal{D}$'s advantage:

$$\Pr[\mathsf{E}^2_{\mathcal{A}}] - \tfrac{1}{2} \leq \mathsf{Adv}^{s\text{-IND-}k\mathsf{CPA}}_{\mathsf{Ch},\mathcal{D}}.$$

This concludes the proof; combining the intermediate advantage bounds yields the overall bound stated in the theorem. □

12.4 Generic Construction of Multi-key Channels from AEAD and PRFs

In this section we construct a generic (deterministic) multi-key channel $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ from on a nonce-based AEAD scheme (cf. Section 9.2) and a pseudorandom function (cf. Section 2.2.1). We then prove that our construction provides the strongest security notions for both confidentiality and integrity in our model, namely indistinguishability under multi-key chosen-ciphertext attacks

$\mathcal{D}^{\mathcal{A},\mathcal{O}_{\mathsf{LoR}},[\mathcal{O}_{\mathsf{Update}}]_{k=\mathsf{mk}},[\mathcal{O}_{\mathsf{Reveal}}]_{s\in\{\mathsf{ki},\mathsf{fski}\}},[\mathcal{O}_{\mathsf{Corrupt}}]_{s\in\{\mathsf{fs},\mathsf{fski}\}}}(1^\lambda)$:

$(\mathsf{st}_S, \mathsf{st}_R) \leftarrow \mathsf{StateGen}(1^\lambda)$
$t_S \leftarrow 0,\ t_R \leftarrow 0,\ i_0 \leftarrow 0,\ j_0 \leftarrow 0$
$\mathsf{sync} \leftarrow 1$
$t_{corr} \leftarrow +\infty,\ \mathit{Rev} \leftarrow \emptyset$
$(t, b) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^*_{\mathsf{LoR}},\mathcal{O}^*_{\mathsf{Recv}},[\mathcal{O}^*_{\mathsf{Update}}]_{k=\mathsf{mk}},[\mathcal{O}^*_{\mathsf{Reveal}}]_{s\in\{\mathsf{ki},\mathsf{fski}\}},[\mathcal{O}^*_{\mathsf{Corrupt}}]_{s\in\{\mathsf{fs},\mathsf{fski}\}}}(1^\lambda)$
if $t > \max(t_S, t_R)$ then
  return 0
return $((b_t = b) \wedge (t \notin \mathit{Rev}) \wedge (t < t_{corr}))$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{LoR}}(m_0, m_1)$:
if $|m_0| \neq |m_1|$ then
  return $\lightning$ to $\mathcal{A}$
$i_{t_S} \leftarrow i_{t_S} + 1$
$\mathbf{C}_S[t_S][i_{t_S}] \leftarrow \mathcal{O}_{\mathsf{LoR}}(m_0, m_1)$
if $t_R > t_S$ and $t_S \notin \mathit{Rev}$ then
  $\mathsf{sync} \leftarrow 0$
return $\mathbf{C}_S[t_S][i_{t_S}]$ to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Update}}(\mathit{role})$:
$\mathcal{O}_{\mathsf{Update}}(\mathit{role})$
if $\mathit{role} = R$ and $t_S \geq t_R$ and $j_{t_R} < i_{t_R}$ and $t_R \notin \mathit{Rev}$ then
  $\mathsf{sync} \leftarrow 0$
$t_{\mathit{role}} \leftarrow t_{\mathit{role}} + 1$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Reveal}}(t, \mathit{role})$:
if $t \leq t_{\mathit{role}}$
  $\mathit{Rev} \leftarrow \mathit{Rev} \cup \{t\}$
return $\mathcal{O}_{\mathsf{Reveal}}(t, \mathit{role})$ to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Corrupt}}(\mathit{role})$:
if $t_{corr} = +\infty$ then
  $t_{corr} \leftarrow t_{\mathit{role}}$
return $\mathcal{O}_{\mathsf{Corrupt}}(\mathit{role})$ to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Recv}}(c)$:
$j_{t_R} \leftarrow j_{t_R} + 1$
if ($t_R > t_S$ or $j_{t_R} > i_{t_R}$ or $c \neq \mathbf{C}_S[t_R][j_{t_R}]$) and $t_R \notin \mathit{Rev}$ then
  $\mathsf{sync} \leftarrow 0$
if $\mathsf{sync} = 0$ then
  if $t_R \notin \mathit{Rev}$ and $t_R < t_{corr}$ then
    $e \leftarrow \mathsf{Pred}(\mathbf{C}_S, \mathbf{C}_R, c)$
    $\mathbf{C}_R[t_R][j_{t_R}] \leftarrow c$
    return e to $\mathcal{A}$
  else if $t_R \in \mathit{Rev}$ then
    $(\mathsf{st}^{\mathit{begin}}_{R,t_R}, \mathsf{K}_{t_R}) \leftarrow \mathcal{O}_{\mathsf{Reveal}}(t_R, R)$
  else
    $(\mathsf{st}^{\mathit{begin}}_{R,t_R}, \mathsf{msk}_{t_R}) \leftarrow \mathcal{O}_{\mathsf{Corrupt}}(R)$
    $\mathsf{K}_{t_R} \leftarrow \mathsf{KeyDerive}(\mathsf{msk}_{t_R})$
  $(\mathsf{st}'_R, \mathbf{m}) \leftarrow \mathsf{Recv}(\mathsf{st}^{\mathit{begin}}_{R,t_R}, \mathsf{K}_{t_R}, \mathbf{C}_R[t_R])$
  $(\mathsf{st}''_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}'_R, \mathsf{K}_{t_R}, c)$
  $\mathbf{C}_R[t_R][j_{t_R}] \leftarrow c$
  return m to $\mathcal{A}$
else
  return $\lightning$ to $\mathcal{A}$

Figure 12.6: Simulation of $\mathsf{E}^2_{\mathcal{A}}$ by the s-IND-kCPA adversary $\mathcal{D}$ in the proof of Theorem 12.7.

and multi-key integrity of ciphertexts, both with forward security and phase-key insulation (fski-IND-mkCCA and fski-INT-mkCTXT).

Our generic construction $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv}, \mathsf{Update})$ is defined via the algorithms given in Figure 12.7. It uses a nonce-based AEAD scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K} = \{0,1\}^\lambda$, message and ciphertext space $\{0,1\}^*$, nonce space $\{0,1\}^n$, associated data space $\{0,1\}^*$, and an error symbol $\perp$. Furthermore, our construction employs a pseudorandom function $f\colon \{0,1\}^\lambda \times \{0,1\} \to \{0,1\}^\lambda$.

Our construction supports a maximum number of $\mathsf{maxmsg} = 2^n$ messages per phase, where n is the AEAD nonce length. The master-secret-key and phase-key space in our construction are equal to the AEAD and PRF key space, $\mathcal{MSK} = \mathcal{K} = \{0,1\}^\lambda$. The error space $\{\perp, \perp'\}$ consists of the error symbol $\perp$ of the AEAD scheme and a second symbol $\perp'$ indicating exceedance of maxmsg. The sending and receiving state space is $\mathcal{S}_S = \mathcal{S}_R = \mathbb{N} \times \mathbb{N}^* \times \{0,1\}$, encoding a message sequence number, a list of the message counts in all previous phases, and a failure flag indicating a previously occurred error.

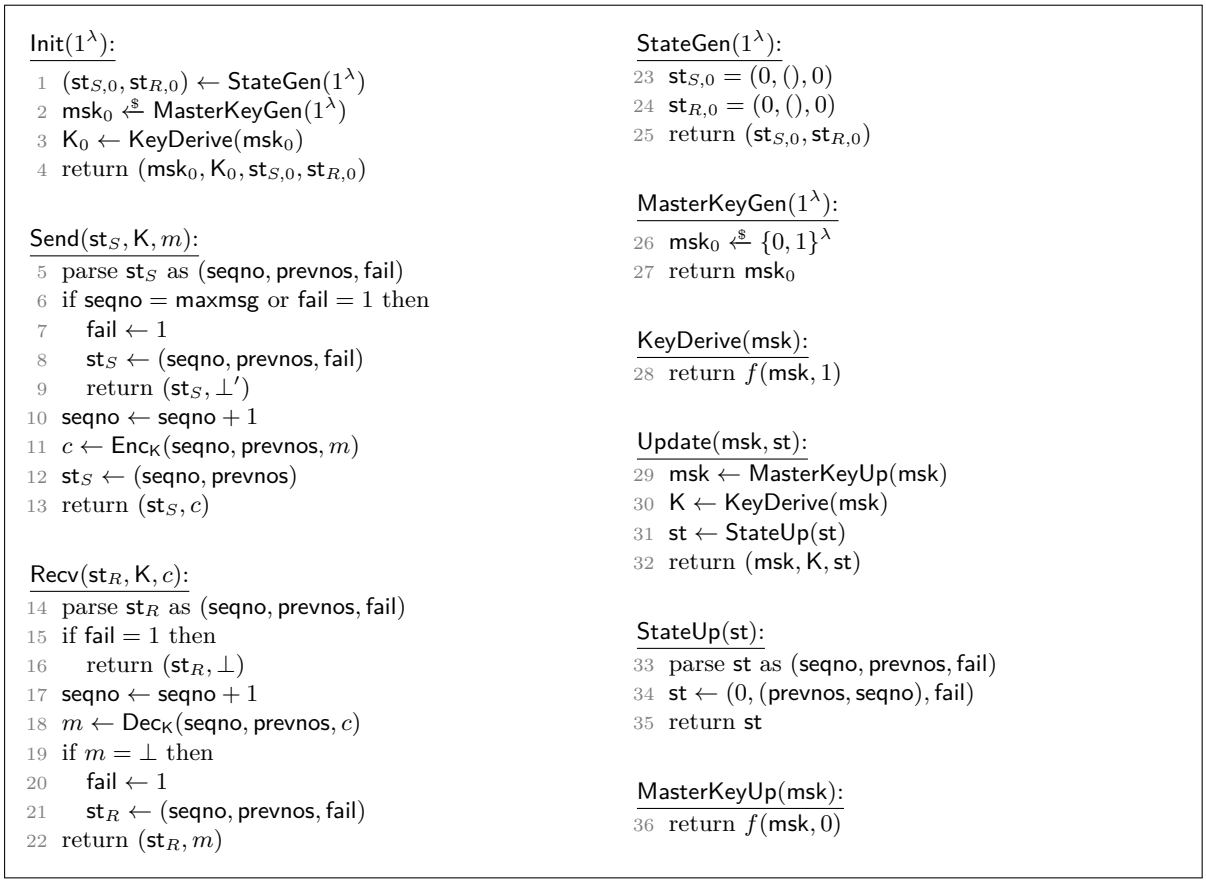


Figure 12.7: Our generic construction of a deterministic multi-key channel $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv}, \mathsf{Update})$ based on a nonce-based authenticated encryption with associated data scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ and a pseudorandom function $f\colon \{0,1\}^\lambda \times \{0,1\} \to \{0,1\}^\lambda$.

On a high level, $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ derives master secret and phase keys via the (domain-separated) PRF f, following the TLS 1.3 key schedule design and employing an established technique to ensure forward security and separation of the keys derived; see, e.g., [BY03]. For encryption, it ensures reorder protection via a sequence number used as nonce. It further authenticates the number of messages seen in previous phases via the associated data field, borrowing established concepts from distributed computing to ensure causality [Lam78].[65]

Construction 12.8 (AEAD- and PRF-based construction $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$)**.** *Consider a nonce-based AEAD scheme* $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ *with key space* $\mathcal{K} = \{0,1\}^\lambda$*, nonce space* $\{0,1\}^n$*, associated data space* $\{0,1\}^*$*, and error symbol* $\perp$*, and a pseudorandom function* $f\colon \{0,1\}^\lambda \times \{0,1\} \to \{0,1\}^\lambda$*. We define* $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv}, \mathsf{Update})$ *to be the multi-key channel algorithmically specified in Figure 12.7 and described in detail below.*

- The Init algorithm uses StateGen to initialize the sending and receiving states as tuples containing a message sequence number seqno $= 0$, a list of the number of messages sent in all previous phases prevnos $= ()$, and a failure flag fail $= 0$. Via MasterKeyGen, the Init

[65] Note that, for a more efficient construction, one can get similar authenticity guarantees by storing a chained hash value of the number of messages received in previous phases using a collision-resistant hash function. For the sake of simplicity we omit this hash-chain optimization here and focus on demonstrating the feasibility of our security notions.

algorithm then samples an initial master secret key $\mathsf{msk}_0 \xleftarrow{\$} \{0,1\}^\lambda$ uniformly at random. Finally it derives the initial phase key $\mathsf{K}_0 \leftarrow f(\mathsf{msk}_0, 1)$ via $\mathsf{KeyDerive}$ as the output of the PRF f keyed with the initial master secret key and on input 1.

- The Send algorithm immediately outputs the error symbol $\perp'$ in case the maximum number $\mathsf{maxmsg} = 2^n$ of messages has been reached in this or a prior call (indicated by $\mathsf{fail} = 1$). Otherwise, it increases the message sequence number in its state by one. It then invokes the deterministic AEAD encryption algorithm on the message m to obtain the ciphertext c. Here, the sequence number is used as the nonce $N = \mathsf{seqno}$ and the previous phases' message count is authenticated through the associated data field $ad = \mathsf{prevnos}$. The output of Send is the new state and the ciphertext c.

- The Recv algorithm immediately outputs an error $\perp$ in case the failure flag has been set ($\mathsf{fail} = 1$) in an earlier invocation, indicating that a previous AEAD decryption algorithm has failed. Otherwise it increases the message sequence number contained in the receiving state by one. It then uses the nonce $N = \mathsf{seqno}$ and associated data $\mathsf{prevnos}$ in the AEAD decryption algorithm on the ciphertext c to obtain m. In case the decryption fails and $m = \perp$, the failure flag is set to 1. The output of Recv is the new state and the message (or error) m.

- The Update algorithm uses $\mathsf{StateUp}$ to reset the new message sequence number to 0, and appends the previous message sequence number to the list of previous phases' message counts, i.e., $\mathsf{prevnos} \leftarrow (\mathsf{prevnos}, \mathsf{seqno})$. Then it invokes $\mathsf{MasterKeyUp}$ to derive a new master secret key as the output of f keyed with the previous master secret key and on input 0. Finally, it uses $\mathsf{KeyDerive}$ to compute a new phase key from the new master secret key.

Correctness. Correctness of our $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ construction follows immediately from correctness of the underlying AEAD scheme. In particular, observe that both receiver and sender compute their master secret and phase keys via the same, deterministic key schedule. Moreover, whenever both sides process the same number—not exceeding maxmsg—of messages per phase (as is a precondition in the correctness definition), they will also use the same associated data values for encryption and decryption, thus rendering the receiver to derive the correct messages as required.

Remark 12.9. At first glance, it might seem counter-intuitive that the sequence number in our $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ construction is reset to 0 at the start of a new phase. Would it not be more natural to have the sequence number running over all phases in order to ensure at the start of a phase that all messages of the previous phase were received, and to prevent reordering of messages across phases?

As surfaced by Fournet and the miTLS [miT] team in the discussion around TLS 1.3 [Fou15], this approach would however enable truncation attacks if the leakage of phase keys is considered in the security definition, as we do for phase-key insulation.[66] If sequence numbers are continued, an adversary holding the key of some phase t can truncate a prefix of the messages (with sequence numbers $i, \ldots, i+j$) in phase $t+1$ by providing the receiver with $j+1$ self-generated messages at the end of t. Dropping the first $j+1$ messages in phase $t+1$, the receiver's sequence number matches again the one of the sender (for message $i+j+1$), so the truncation would go unnoticed. Resetting the sequence numbers to 0 when switching phases prevents this attack, though additional care needs to be taken to prevent suffix truncation at the end of a

[66] In our framework, the weakest integrity property broken through this attack is phase-key–insulated integrity of plaintexts ($\mathsf{ki\text{-}INT\text{-}mkPTXT}$).

phase. In our construction, we ensure the latter through authenticating the number of messages sent in all previous phases. We note that this mechanism would even allow to not reset the sequence number, but we decided to keep the reset in order to stay closer to the channel design of TLS 1.3 [Res18] (cf. the discussion in Section 12.4.2).

12.4.1 Security Analysis

We now show that our generic $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ construction achieves the strongest multi-key security notions for confidentiality and integrity, namely forward-secure and phase-key–insulated indistinguishability under multi-key chosen-ciphertext attacks (fski-IND-mkCCA) and integrity of ciphertexts (fski-INT-mkCTXT). For proving the former notion we proceed via first showing the corresponding CPA confidentiality variant as well as that our construction provides error predictability (for multiple keys and with forward security and phase-key insulation), and then leverage our generic composition theorem (Theorem 12.7). Our results hold under the assumption that the underlying nonce-based AEAD scheme AEAD provides IND-CPA confidentiality and INT-CTXT integrity (cf. Section 9.2.1), as well as that the employed pseudorandom function f meets the standard notion of PRF security (cf. Section 2.2.1).

We begin with the proof of multi-key chosen-plaintext confidentiality with forward security and phase-key insulation.

Theorem 12.10 (fski-IND-mkCPA security of $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$)**.** *The multi-key channel* $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ *from Construction 12.8 provides forward-secure and phase-key–insulated indistinguishability under multi-key chosen-plaintext attacks (*fski-IND-mkCPA*) if the employed authenticated encryption with associated data scheme* AEAD *provides indistinguishability under chosen-plaintext attacks (*IND-CPA*) and the employed pseudorandom function f is* PRF*-secure.*

Formally, for every efficient fski-IND-mkCPA *adversary $\mathcal{A}$ against* $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ *there exists an efficient* PRF-sec *adversary $\mathcal{B}$ against f and* IND-CPA *adversary $\mathcal{C}$ against* AEAD *such that*

$$\mathsf{Adv}^{\mathsf{fski\text{-}IND\text{-}mkCPA}}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathcal{A}}(\lambda) \leq n_t \cdot \left(n_t \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{f,\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AEAD},\mathcal{C}}(\lambda)\right),$$

where $n_t = \max(t_S, t_R) + 1$ is the maximum number of phases active in the fski-IND-mkCPA *experiment (plus one).*

Proof. Our proof proceeds in three steps. First, we guess the phase t that the adversary $\mathcal{A}$ will pick as its challenge phase (out of the at most n_t active phases in the experiment). Aborting in case of a wrong guess induces a loss in the advantage of $\mathcal{A}$ by at most a factor of n_t. Denoting with $\mathsf{E}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t}_{\mathcal{A}}$ the resulting experiment we hence have that

$$\mathsf{Adv}^{\mathsf{fski\text{-}IND\text{-}mkCPA}}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathcal{A}}(\lambda) \leq n_t \cdot \mathsf{Adv}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathcal{A}}(\lambda).$$

From now on, we can assume that $\mathcal{A}$ will issue its guess for the challenge phase t which we furthermore know in advance.

In the second step, we gradually replace all derived master secret keys up to (including) msk_{t+1} of phase $t+1$ as well as the phase keys derived up to (including) K_t of phase t with independent random values. Let $\mathsf{E}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$i}_{\mathcal{A}}$ denote the $\mathsf{E}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t}_{\mathcal{A}}$ experiment with the modification that the master secret keys in phases 0 to i as well as the phase keys in phases 0 to $i-1$ are chosen independently at random as $\mathsf{msk}_0, \ldots, \mathsf{msk}_i, \mathsf{K}_0, \ldots, \mathsf{K}_{i-1} \xleftarrow{\$} \{0,1\}^\lambda$. In particular, the $\mathsf{E}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$0}_{\mathcal{A}}$ experiment equals $\mathsf{E}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t}_{\mathcal{A}}$ where only the initial master secret key msk_0 is randomly chosen (as defined by $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$). Furthermore, $\mathsf{E}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$t+1}_{\mathcal{A}}$ denotes an experiment where all master secret keys up to (including) msk_{t+1} (of phase $t+1$)

and all phase keys up to (including) K_t (of phase t) are chosen uniformly at random; i.e., in particular the key K_t of the challenge phase t picked by $\mathcal{A}$.

We now bound the difference in advantage between two games $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$i}$ and $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$i+1}$ (for some $i \in \{0,\ldots,t\}$) by the advantage of an algorithm $\mathcal{B}$ against the PRF security of f. When simulating the $\mathsf{fski\text{-}IND\text{-}mkCPA}$ experiment for $\mathcal{A}$, algorithm $\mathcal{B}$ picks an index $i \in \{0,\ldots,t\}$ at random and follows the experiment and construction description, but samples all master secret keys msk_j for $j \leq i$ and all phase keys K_j for $j \leq i-1$ uniformly at random from $\{0,1\}^\lambda$. In the moment $\mathcal{B}$ is supposed to derive $\mathsf{msk}_{i+1} \leftarrow f(\mathsf{msk}_i, 1)$ or $\mathsf{K}_i \leftarrow f(\mathsf{msk}_i, 0)$, it queries the values 1 resp. 0 to its PRF oracle. For all following master secret and phase keys, $\mathcal{B}$ follows the $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ and derives them via f as specified. Recall that the PRF keys used for these derivations are independent of the implicit PRF key msk_i used in the PRF oracle. Furthermore, $\mathcal{B}$ never has to provide the implicit PRF key msk_i to $\mathcal{A}$ through an $\mathcal{O}_{\mathsf{Corrupt}}$ call, as $i \leq t < t_{corr}$. When $\mathcal{A}$ stops and outputs its guess b, $\mathcal{B}$ also stops and outputs 1 if the guess was correct (i.e., $b = b_t$) and 0 otherwise. Denote by $\mathcal{B}^i$ the reduction $\mathcal{B}$ picking index i. Observe that $\mathcal{B}^i$ correctly simulates experiment $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$i}$ for $\mathcal{A}$ in case its PRF oracle computes the real PRF f; otherwise it simulates $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$i+1}$ as the (random-function) PRF oracle outputs two independent random values on inputs 1 and 0. Furthermore, any difference in $\mathcal{A}$'s output behavior between the two experiments translates into a difference in $\mathcal{B}^i$'s output in the PRF security game. Hence, we can bound the former as

$$\left|\mathsf{Adv}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$i}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$i+1}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathcal{A}}(\lambda)\right| \leq \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{f,\mathcal{B}^i}(\lambda).$$

Via a hybrid argument, we can therefore infer that the advantage difference introduced when switching from $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t} = \mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$0}$ to $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$t+1}$ is bounded as follows (keeping in mind that $t+1 \leq n_t$):

$$\mathsf{Adv}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathcal{A}}(\lambda) \leq n_t \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{f,\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$t+1}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathcal{A}}(\lambda).$$

In the third and last step, we argue that the advantage of $\mathcal{A}$ in game $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$t+1}$ can be bounded by the $\mathsf{IND\text{-}CPA}$ security of the employed AEAD scheme. Consider the following reduction $\mathcal{C}$. To simulate $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$t+1}$ for $\mathcal{A}$, algorithm $\mathcal{C}$ carries out all steps in the experiment and construction algorithms on its own, except for parts of the operations in the sending oracle $\mathcal{O}_{\mathsf{Send}}$ in phase t. (Recall that, as we are proving CPA security, there is no receiving oracle available to $\mathcal{A}$.) Particularly, $\mathcal{C}$ picks all challenge bits b_i except for $i = t$ on its own at random. In phase t (that is, starting from the t-th call and up to the $t+1$-th call of the $\mathcal{O}_{\mathsf{Update}}$ oracle on input $role = S$), $\mathcal{C}$ does not pick b_t and also does not perform the $\mathsf{Enc}_{\mathsf{K}_t}$ operation within the Send algorithm of $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ on its own. Instead of computing $c \leftarrow \mathsf{Enc}_{\mathsf{K}_t}(\mathsf{seqno}, \mathsf{prevnos}, m_{b_t})$ itself, it queries the encryption oracle of its $\mathsf{IND\text{-}CPA}$ game on $N = \mathsf{seqno}$ and $ad = \mathsf{prevnos}$ together with m_0 and m_1 (as provided by $\mathcal{A}$ to $\mathcal{O}_{\mathsf{Send}}$) and uses the result as ciphertext value c. Note that $\mathcal{C}$ never exceeds the nonce space of the AEAD scheme as Send ensures that $\mathsf{seqno} \leq \mathsf{maxmsg}$. Finally, when the adversary $\mathcal{A}$ outputs its guess b for phase t, $\mathcal{C}$ also outputs b as its own guess.

Observe first of all that $\mathcal{C}$ correctly simulates $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$t+1}$ for $\mathcal{A}$. For simulating the sending oracle $\mathcal{O}_{\mathsf{Send}}$, $\mathcal{C}$ holds the keys $\mathsf{K}_{t'}$ for all phases $t' \neq t$ itself and can hence execute the Send algorithm as specified. In the $\mathcal{O}_{\mathsf{Update}}$ oracle, $\mathcal{C}$ simply derives master secret and phase keys as specified for $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$t+1}$, i.e., it chooses them independently at random up to msk_{t+1}, resp. K_t, and derives all further keys through f. As t is the challenge phase for which $\mathcal{A}$ outputs its guess, we furthermore know that (a successful adversary) $\mathcal{A}$ will neither issue an $\mathcal{O}_{\mathsf{Reveal}}$ query on t nor an $\mathcal{O}_{\mathsf{Corrupt}}$ query such that $t_{corr} < t+1$. In particular, the phase keys $\mathsf{K}_{t'}$ (for $t \neq t'$) as well as a potentially corrupted master secret key $\mathsf{msk}_{t'}$ (for $t' > t$) that $\mathcal{A}$ obtains

in this way are completely independent of the phase key in phase t. It is hence sound that $\mathcal{C}$ does not employ a self-chosen random key K_t in this challenge phase but implicitly sets K_t to the random key chosen in the IND-CPA game for the AEAD scheme. Moreover, by invoking its IND-CPA encryption oracle within the representation of the Send algorithm in phase t, $\mathcal{C}$ also implicitly sets the challenge bit b_t in that phase to the one in the IND-CPA game. Outputting the same bit as $\mathcal{A}$ thus makes $\mathcal{C}$ correctly determine the IND-CPA challenge bit if $\mathcal{A}$ correctly guesses b_t and hence

$$\mathsf{Adv}^{\mathsf{fski\text{-}IND\text{-}mkCPA},t,\$t+1}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AEAD},\mathcal{C}}(\lambda).$$

This concludes the proof. Combining the intermediate advantage bounds yields the overall bound. □

We now turn to the multi-key integrity of ciphertexts with forward security and phase-key insulation of $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$.

Theorem 12.11 (fski-INT-mkCTXT security of $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$)**.** *The multi-key channel* $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ *from Construction 12.8 provides forward-secure and phase-key–insulated multi-key integrity of ciphertexts (*fski-INT-mkCTXT*) if the employed authenticated encryption with associated data scheme* AEAD *provides integrity of ciphertexts (*INT-CTXT*) and the employed pseudorandom function f is* PRF*-secure.*

Formally, for every efficient fski-INT-mkCTXT *adversary $\mathcal{A}$ against* $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ *there exists an efficient* PRF-sec *adversary $\mathcal{B}$ against f and* INT-CTXT *adversary $\mathcal{C}$ against* AEAD *such that*

$$\mathsf{Adv}^{\mathsf{fski\text{-}INT\text{-}mkCTXT}}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathcal{A}}(\lambda) \leq n_t \cdot \left(n_t \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{f,\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD},\mathcal{C}}(\lambda) \right),$$

where $n_t = \max(t_S, t_R) + 1$ is the maximum number of phases active in the fski-INT-mkCTXT *experiment (plus one).*

Proof. The first two steps of this proof follow closely those of the proof of fski-IND-mkCPA security of $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ (cf. Theorem 12.10). We first guess a "challenge" phase t (among the total n_t number of phases) and abort on an incorrect guess. Recall that in the integrity experiment (cf. Figure 12.3) the adversary $\mathcal{A}$ provides no output; in particular it does not have to commit on a challenge phase like in the confidentiality experiment. Nevertheless, in this proof we define the challenge phase t for the experiment to be the value t_R in the moment when, in the $\mathcal{O}_{\mathsf{Recv}}$ oracle, within the condition check in Line 22 of the experiment for the first time the conditions $\mathsf{sync} = 0$, $t_R \notin \mathit{Rev}$, and $t_R < t_{corr}$ all evaluate to true. In the following, we refer to this moment as the "challenge moment," to the corresponding $\mathcal{O}_{\mathsf{Recv}}$ oracle call as the "challenge oracle call," and to the input ciphertext c in this call as the "challenge ciphertext."

Second, we follow the same hybrid step as in the proof of Theorem 12.10 to replace the master secret and phase keys up to msk_{t+1} and K_t with independent random values. Combined with the first step and using the same notation as in the confidentiality proof, this yields the following bound:

$$\mathsf{Adv}^{\mathsf{fski\text{-}INT\text{-}mkCTXT}}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathcal{A}}(\lambda) \leq n_t \cdot \left(n_t \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{f,\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{fski\text{-}INT\text{-}mkCTXT},t,\$t+1}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathcal{A}}(\lambda) \right).$$

For the final step, it remains to show that $\mathcal{A}$'s advantage in the modified experiment $\mathsf{E}^{\mathsf{fski\text{-}INT\text{-}mkCTXT},t,\$t+1}_{\mathcal{A}}$ can be bounded by the advantage of a reduction $\mathcal{C}$ to the INT-CTXT security of AEAD. To this end, first of all observe that the construction $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ (cf. Figure 12.7) will reject (and output $\perp$ on) any further ciphertext received after, within Recv, the AEAD decryption algorithm Dec output the error symbol $\perp$ for the first time. Hence in particular if the adversary $\mathcal{A}$ does not make the winning flag win set to 1 in the challenge moment, it cannot win the game anymore later. We can therefore deduce that a successful

adversary will win in the challenge moment, i.e., the fourth condition in Line 22, $m \notin \mathcal{E}$ also evaluates to true, i.e., $m \neq \perp$.

Our reduction $\mathcal{C}$ to the integrity (INT-CTXT) of the AEAD scheme proceeds as follows. In the beginning, $\mathcal{C}$ picks all phase keys up to including K_{t-1} as well as the master secret key msk_{t+1} uniformly at random on its own. Algorithm $\mathcal{C}$ then simulates the $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}INT\text{-}mkCTXT},t,\$t+1}$ for $\mathcal{A}$ by performing operations for all phases except phase t itself using the chosen keys. In the challenge phase t, $\mathcal{C}$, instead of computing $\mathsf{Enc}_{\mathsf{K}_t}$ and $\mathsf{Dec}_{\mathsf{K}_t}$ on its own, it uses its encryption and decryption oracles in the INT-CTXT game to perform these operations during sending and receiving. Note that, due to the definition of the challenge phase t, $t \notin \mathit{Rev}$ and $t < t_{corr}$. Hence, $\mathcal{A}$ does not ask $\mathsf{Reveal}(t, \mathit{role})$ nor does it corrupt a master secret key for a phase $t' \leq t$. Therefore, $\mathcal{C}$ also does not have to be able to respond to those queries. At last, $\mathcal{C}$ follows the $\mathcal{O}_{\mathsf{Update}}$ specification to, if necessary, derive master secret key msk_{t+2} and following as well as phase keys K_{t+1} and following. In the moment $\mathcal{C}$ would set the winning flag $\mathsf{win} \leftarrow 1$ during an oracle call $\mathcal{O}_{\mathsf{Recv}}(c)$ of $\mathcal{A}$ for some ciphertext c in its simulation, it stops and outputs c along with the nonce N and associated data ad as used in the $\mathcal{O}_{\mathsf{Recv}}$ oracle as its forgery in the INT-CTXT game. Overall, $\mathcal{C}$ provides a sound simulation of $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}INT\text{-}mkCTXT},t,\$t+1}$ for $\mathcal{A}$.

We finally have to argue that $\mathcal{C}$ wins in the INT-CTXT game if $\mathcal{A}$ does in $\mathsf{E}_{\mathcal{A}}^{\mathsf{fski\text{-}INT\text{-}mkCTXT},t,\$t+1}$. For this purpose, we separately consider the case that synchronization was lost ($\mathsf{sync} \leftarrow 0$) in the same $\mathcal{O}_{\mathsf{Recv}}$ call that leads to the challenge moment, and the case that it was lost earlier. In the first case, synchronization loss requires one of the following conditions to be true in Line 21:

- $t_R > t_S$ (the receiver's phase t_R is ahead of the sender's phase t_S in the challenge moment).
  In this case, we are ensured that the challenge ciphertext c cannot have been the output of the INT-CTXT encryption oracle, as no such call was made yet (recall that the encryption oracle is only invoked for sending in phase $t_R = t$). Hence, (N, ad, c) output by $\mathcal{C}$ is a valid forgery and makes $\mathcal{C}$ win as $m \neq \perp$.

- $j_{t_R} > i_{t_R}$ (the receiver obtained more ciphertexts in phase t_R as have been sent).
  In this case, the INT-CTXT encryption oracle was not called on sequence number $\mathsf{seqno} = j_{t_R}$ (i.e., the output nonce $N = \mathsf{seqno}$ is fresh), so again (N, ad, c) is a valid forgery making $\mathcal{C}$ win.

- $c \neq \mathbf{C}[t_R][j_{t_R}]$ (the received and sent ciphertexts mismatch).
  Due to the ciphertext mismatch, the output of the INT-CTXT encryption oracle for sequence number $\mathsf{seqno} = j_{t_R}$ (as nonce N) must have been different from c. The latter hence again is a valid forgery that qualifies $\mathcal{C}$ for winning.

We now treat the case that synchronization was lost before the challenge oracle call to $\mathcal{O}_{\mathsf{Recv}}$. There are three positions in the integrity experiment where synchronization can be lost (cf. Figure 12.3) which we consider separately:

- Line 11 in $\mathcal{O}_{\mathsf{Send}}$: As $t_R > t_S$, we know that through this $\mathcal{O}_{\mathsf{Send}}$ call the sending counter will stay ahead of the receiving counter for the current sender's phase t_S throughout the experiment (i.e., $i_{t_S} > j_{t_S}$). Otherwise, we would have received at least one additional ciphertext c beyond the sent ciphertexts in phase t_S. As $t_S \notin \mathit{Rev}$ and $t_S < t_{corr}$, this would have triggered synchronization to be lost in the according $\mathcal{O}_{\mathsf{Recv}}$ call processing this ciphertext c (and hence the current call would not be the one where synchronization is lost).

  As the send counter i_{t_S} of phase t_S is accumulated within $\mathsf{prevnos}$ used as associated data ad, the ad value in the $\mathcal{O}_{\mathsf{Recv}}$ oracle when processing the challenge ciphertext will necessarily contain a different ciphertext count for t_S as the one used when encrypting

the corresponding ciphertext on the sender's side. The associated data used in $\mathcal{O}_{\mathsf{Recv}}$ was hence never sent to the INT-CTXT encryption oracle and hence (N, ad, c) output by $\mathcal{C}$ constitutes a valid forgery.

- Line 21 in $\mathcal{O}_{\mathsf{Recv}}$: If synchronization is lost in Line 21 without this oracle call being the challenge call, the condition in Line 20 must evaluate to true while the condition in Line 22 for this call must evaluate to false. Careful inspection shows that this necessitates that $m \in \mathcal{E}$, i.e., $m = \perp$ in this call. As discussed above, the construction $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ will only output further error messages $\perp$ after the first error is output, leaving $\mathcal{A}$ with no chance to win from this point on. Hence, we can deduce that, for a successful adversary, synchronization is not lost in Line 21 of an $\mathcal{O}_{\mathsf{Recv}}$ call earlier than the challenge one.

- Line 27 in $\mathcal{O}_{\mathsf{Update}}$: A synchronization loss in this line means the receiver cannot have obtained all sent ciphertexts in phase t_R. Note moreover that $\mathcal{O}_{\mathsf{Recv}}$ cannot be called in phase t_R anymore, as the receiver's phase is increased immediately after Line 27 in the $\mathcal{O}_{\mathsf{Update}}$ call. It will hence use a different ciphertext count for this phase within the accumulated prevnos value contained in the associated data field for all follow-up received ciphertexts. As in the argument for Line 11, there will in particular be no INT-CTXT encryption oracle call made with the associated data used to encrypt the challenge ciphertext c, making the output (N, ad, c) of $\mathcal{C}$ a valid forgery.

In summary, if $\mathcal{A}$ wins in Line 22 in the challenge moment (which is the only moment it can win at all), the corresponding challenge ciphertext c along with the nonce and associated data field used in the challenge oracle $\mathcal{O}_{\mathsf{Recv}}$ constitutes a valid forgery in the INT-CTXT game, which $\mathcal{C}$ outputs. Hence,

$$\mathsf{Adv}^{\mathsf{fski\text{-}INT\text{-}mkCTXT},t,\$t+1}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD},\mathcal{C}}(\lambda),$$

concluding the proof. □

Finally, we show that our $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ construction provides multi-key error predictability with forward security and phase-key insulation (fski-mkERR-PRE).

Theorem 12.12 (fski-mkERR-PRE security of $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$)**.** *The multi-key channel* $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ *from Construction 12.8 provides forward-secure and phase-key–insulated multi-key error predictability (*fski-mkERR-PRE*) with respect to the error predictor* Pred *given in the proof of the theorem.*

Formally, for every efficient fski-mkERR-PRE *adversary* $\mathcal{A}$ *against* $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$*,*

$$\mathsf{Adv}^{\mathsf{fski\text{-}mkERR\text{-}PRE}}_{\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}},\mathsf{Pred},\mathcal{A}}(\lambda) = 0.$$

Proof. Consider the error predictor Pred which always output the AEAD error symbol $\perp$.

In order for $\mathcal{A}$ to win the error predictability experiment fski-mkERR-PRE (cf. Figure 12.5, Line 11), it must make the Recv algorithm output an error message ($m \in \mathcal{E}$) which differs from the predictor's output ($m \neq \mathsf{Pred}(\mathbf{C}_S, \mathbf{C}_R, c)$). However, by construction the only error symbol $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ ever outputs in Recv is $\perp$, hence the given predictor will never differ from error messages of $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ and hence $\mathcal{A}$ cannot win. □

As $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ provides security in the senses of fski-IND-mkCPA, fski-INT-mkCTXT, and fski-mkERR-PRE, we can finally leverage our generic composition result from Theorem 12.7 to conclude that it also achieves strong confidentiality in the sense of fski-IND-mkCCA.

Corollary 12.13 (fski-IND-mkCCA security of $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$)**.** *The multi-key channel* $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ *from Construction 12.8 provides forward-secure and phase-key–insulated indistinguishability under multi-key chosen-ciphertext attacks (*fski-IND-mkCCA*) if the employed authenticated encryption with associated data scheme* AEAD *provides indistinguishability under chosen-plaintext attacks (*IND-CPA*) as well as integrity of ciphertexts (*INT-CTXT*) and the employed pseudorandom function f is* PRF*-secure.*

12.4.2 Comparison to the TLS 1.3 Record Protocol

Our notion of multi-key channels is particularly inspired by the ongoing developments of the upcoming Transport Layer Security (TLS) protocol version 1.3 [Res18]. It is hence insightful to compare our generic construction with the design of the TLS 1.3 record protocol (cf. [Res18, Section 5]).

First of all note that, in contrast to previous TLS versions, TLS 1.3 mandates the use of AEAD schemes as encryption and authentication mechanisms for the record protocol. It follows the basic secure-channel design principle to include a sequence number for protecting against reordering attacks; as in our construction. Both in TLS 1.3 and our construction, the sequence number enters the AEAD's nonce field and is reset to 0 at the start of each new phase. Also identically to our construction, the TLS 1.3 record protocol keys are derived via a deterministic key schedule in which, starting from an initial master secret key (denoted `client/server_application_traffic_secret_0` in TLS 1.3) the current phase's key as well as the next phase's master secret key are derived via independent applications of a pseudorandom function (TLS 1.3 uses HMAC [BCK96, KBC97] for this purpose). Beyond enabling key switches to allow secure encryption of large amounts of data, the TLS 1.3 design in particular names forward security (combined with insulation of phase keys) as a security goal [Res18, Appendix E.2]. In this sense, our generic $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ construction is comparatively close to the internal channel design of the TLS 1.3 record protocol in both techniques and security goals.

Still, there are some notable differences between the two designs, both in technical details as well as in the practically achieved security and its underlying assumptions. On the technical side, the TLS 1.3 record protocol additionally includes a content-type field in ciphertexts to enable multiplexing of messages from multiple sources. It also provides a streaming interface to applications, as discussed in Chapter 10, while our model and construction focuses on the simpler setting with an atomic-message interface. Furthermore, TLS 1.3 does not explicitly authenticate the numbers of seen ciphertexts in previous phases (as our construction does via the prevnos field), but instead relies on the authenticated transmission of key update messages. To be precise, key update messages are encoded as a specific control ("post-handshake") message and sent within the data channel. Thereby associated with a sequence number, they serve as an authenticated "end-of-phase indicator" that allows the record protocol to infer in unrevealed phases that all messages in a phase have been correctly received when the key update message arrives.

In contrast, our model does not rely on the authenticity of key updates (an approach worth augmenting our model with in a future work step), but captures settings where key update notifications may be send out-of-band and without being authenticated. Our construction hence cannot rely on key updates as indicators that a phase was gracefully completed, but instead needs to leverage the next uncompromised phase to detect truncations in an earlier phase; Nevertheless, our generic $\mathsf{Ch}_{\mathsf{AEAD\text{-}PRF}}$ scheme serves as proof-of-concept construction that strong confidentiality and integrity can be achieved in the multi-key setting with forward security and phase-key insulation even with unauthenticated, out-of-band key updates.

Chapter 13

Conclusion

The basic cryptographic security of key exchange and secure channel protocols is considered to be well-understood. In this thesis, we studied how advanced security aspects of both protocol types can be formally captured in terms of enhanced cryptographic security models. We treated well-known but more specific properties that have not been captured formally yet, protocol behavior that has not been accurately reflected by the established security notions so far, and recent protocol designs (specifically QUIC [QUI] and TLS 1.3 [Res18]) introducing novel features that go beyond what security models could capture until now.

In Part I, we considered advanced aspects of key exchange protocols. Our main focus has been on novel key exchange designs, specifically QUIC and TLS 1.3, that do not establish a single shared key, as classically considered in key exchange models, but multiple keys with potentially differing security properties. We captured such protocols in our novel security model for *multi-stage key exchange* (MSKE) protocols extending the classical security notions for key exchange in the style of Bellare and Rogaway [BR94]. Our model allowed us to capture multiple keys being derived in a single key exchange protocol execution including, e.g., their possible interdependency and resulting affects on their security, or different security and authentication levels. We applied our model to assess and confirm the (multi-stage) key exchange security of both the QUIC protocol as well as several draft versions and handshake modes of the TLS 1.3 protocol, gaining insights into the cryptographic structure of both protocol designs and providing valuable feedback into their design processes. Upon completion of the IETF standardization process, our model will enable a comprehensive analysis of the final TLS 1.3 handshake design.

Another particularly interesting feature of both QUIC and TLS 1.3 is that both protocols introduce a low-latency, *zero round-trip time* (0-RTT) handshake mode. This mode comes with substantial performance benefits, but also inherent drawbacks in security. We considered the specific issue that 0-RTT data can potentially be replayed by an active adversary and augmented our MSKE security model to capture such replays, enabling a comparison of the 0-RTT techniques deployed in different draft versions of TLS 1.3 as well as in QUIC.

Our last contribution in the realm of key exchange was concerned with a property often and for a long time mentioned in practical protocol designs, but never comprehensively formalized: *key confirmation*, providing assurance that a communication partner accepts the same key. We introduced a formal treatment of key confirmation, establishing two flavors as the communication partners necessarily obtain (slightly) different guarantees depending on which of them terminates first. We then used our formalization to establish key confirmation guarantees for the TLS 1.3 handshake draft. Along the way, our analysis uncovered that key confirmation can not only be obtained from exchanged message authentication codes (as commonly understood), but also already through online signatures.

In Part II, we turned towards secure channel protocols. All cryptographic security models for secure channels so far consider the messages to be send as atomic, undividable units, and likewise the resulting ciphertexts (with [BDPS12] being an exception on the latter). In practice, secure channels however provide applications with a *streaming* interface and fragment messages. Mis-interpreting the resulting security guarantees has led to severe attacks in the past, e.g., on SSH [YL06a] and TLS [DR08]. In order to study the practical security of channels more accurately, we introduced notions (in syntax and security) for stream-based channels. We saw that, along with the inherent complexity of such channels, the security notions turned out to be more complex, too, specifically those considering active attacks. Yet, we were able to lift the generic composition theorem to the stream-based setting, allowing to deduce confidentiality against active attackers from the conceptually simpler and easier to prove notions of passive confidentiality and integrity of ciphertexts. We confirmed that our notions are achievable in a natural construction based on authenticated encryption with associated data (AEAD) which, moreover, is similar to the TLS record protocol and hence enables a validation of that design.

We then turned towards such applications running atop of a streaming channel, yet actually sending atomic messages and hence needing to ensure that original messages are carefully re-assembled to avoid vulnerabilities. We analyzed the common approach in practice to encode atomic messages into the stream of data to be send. To this end, we extended the formalism for channels sending atomic messages over a fragmenting network and established that, under natural conditions, the described "encode-then-stream" approach indeed achieves strong confidentiality and integrity in this setting. Seeing security-critical misuses of streaming interfaces, this raises the practical question if channels should provide both streaming and atomic-message interfaces.

Finally, we studied the novel concept introduced in the TLS 1.3 design to update keys during the lifetime of a channel to enhance its security. Introducing the notion of multi-key channels, we provided formalizations of advanced security properties under partial compromise of the channel keys, as aimed at by TLS 1.3. Our security notions for multi-key channels span a hierarchy that naturally connects to the classical notions for single-key channels. Furthermore, we confirmed that a generic construction from AEAD and pseudorandom functions being comparatively close to the TLS 1.3 channel can indeed achieve the strongest security in our setting. Approaching the TLS 1.3 design even further, a treatment of key update authentication within the channel protocol as well as of multi-key channels in the streaming setting emerge as potential extensions.

Overall, the designs of practical key exchange and secure channel protocols seem to become more and more involved, aiming at further and novel security aspects over the last years and in the foreseeable future. This raises the question to which extent established cryptographic models, e.g., those from this thesis, can be applied (and possibly extended) to capture further established or emerging protocols without introducing too much additional complexity. Secure messaging protocols like Signal [Sig] constitute particularly challenging examples due to the novel security goals (like post-compromise security [CGCG16]) they aim at, resulting in accordingly elaborate designs and key schedules. Our multi-stage key exchange and multi-key channel models may be particularly suited for these settings, capturing potential dependencies and varying security strengths of the derived keys and their sequential usage to encrypt messages. Indeed, the MSKE model has already been applied by others working in that direction [CGCD$^+$17, CGCG$^+$17].

In the end, work in applied cryptography and provable security is striving for better and more accurate security models, in turn pushing the boundaries of practical protocols that can be designed on solid, provably-secure grounds. I hope this thesis can contribute a small share to that big and growing task.

Bibliography

[AB00] Michel Abdalla and Mihir Bellare. Increasing the lifetime of a key: a comparative analysis of the security of re-keying techniques. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 546–559, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany.

[ABD+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15: 22nd Conference on Computer and Communications Security*, pages 5–17, Denver, CO, USA, October 12–16, 2015. ACM Press.

[ABP+13] Nadhem AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. On the security of RC4 in TLS. In Samuel T. King, editor, *Proceedings of the 22th USENIX Security Symposium*, pages 305–320, Washington, DC, USA, August 14–16, 2013. USENIX Association.

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158, San Francisco, CA, USA, April 8–12, 2001. Springer, Heidelberg, Germany.

[ADHP16] Martin R. Albrecht, Jean Paul Degabriele, Torben Brandt Hansen, and Kenneth G. Paterson. A surfeit of SSH cipher suites. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16: 23rd Conference on Computer and Communications Security*, pages 1480–1491, Vienna, Austria, October 24–28, 2016. ACM Press.

[AP13] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[APW09] Martin R. Albrecht, Kenneth G. Paterson, and Gaven J. Watson. Plaintext recovery attacks against SSH. In *2009 IEEE Symposium on Security and Privacy*, pages 16–26, Oakland, CA, USA, May 17–20, 2009. IEEE Computer Society Press.

[AR02] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[BBD+15] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *2015 IEEE Symposium on Security and Privacy*, pages 535–552, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.

[BBF+16] Karthikeyan Bhargavan, Christina Brzuska, Cédric Fournet, Matthew Green, Markulf Kohlweiss, and Santiago Zanella Béguelin. Downgrade resilience in key-exchange protocols. In *2016 IEEE Symposium on Security and Privacy*, pages 506–525, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press.

[BBK17] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy*, pages 483–502, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press.

[BCF+13] Colin Boyd, Cas Cremers, Michele Feltz, Kenneth G. Paterson, Bertram Poettering, and Douglas Stebila. ASICS: Authenticated key exchange security incorporating certification systems. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 381–399, Egham, UK, September 9–13, 2013. Springer, Heidelberg, Germany.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany.

[BCRS09] Elaine Barker, Lily Chen, Andrew Regenscheid, and Miles Smid. SP 800-56B. Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography. NIST Special Publication, National Institute of Standards & Technology, August 2009.

[BCRS13] Elaine Barker, Lily Chen, Allen Roginsky, and Miles Smid. SP 800-56A r2. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. NIST Special Publication, National Institute of Standards & Technology, May 2013.

[BDF+14] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *2014 IEEE Symposium on Security and Privacy*, pages 98–113, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press.

[BDPS12] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. Security of symmetric encryption in the presence of ciphertext fragmentation. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 682–699, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[BDPS14] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. On symmetric encryption with distinguishable decryption failures. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of

Lecture Notes in Computer Science, pages 367–390, Singapore, March 11–13, 2014. Springer, Heidelberg, Germany.

[BF17] Jacqueline Brendel and Marc Fischlin. Zero round-trip time for the extended access control protocol. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *ESORICS 2017: 22nd European Symposium on Research in Computer Security, Part I*, volume 10492 of *Lecture Notes in Computer Science*, pages 297–314, Oslo, Norway, September 11–15, 2017. Springer, Heidelberg, Germany.

[BFGJ17] Jacqueline Brendel, Marc Fischlin, Felix Günther, and Christian Janson. PRF-ODH: Relations, instantiations, and impossibility results. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 651–681, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[BFK+13] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, and Pierre-Yves Strub. Implementing TLS with verified cryptographic security. In *2013 IEEE Symposium on Security and Privacy*, pages 445–459, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[BFK+14] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella Béguelin. Proving the TLS handshake secure (as it is). In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 235–255, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[BFPW07] Alexandra Boldyreva, Marc Fischlin, Adriana Palacio, and Bogdan Warinschi. A closer look at PKI: Security and efficiency. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 458–475, Beijing, China, April 16–20, 2007. Springer, Heidelberg, Germany.

[BFS+13] Christina Brzuska, Marc Fischlin, Nigel P. Smart, Bogdan Warinschi, and Stephen C. Williams. Less is more: relaxed yet composable security notions for key exchange. *Int. J. Inf. Sec.*, 12(4):267–297, 2013.

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 51–62, Chicago, Illinois, USA, October 17–21, 2011. ACM Press.

[BH17] Colin Boyd and Britta Hale. Secure channels and termination: The last word on tls. In *Progress in Cryptology - LATINCRYPT 2017: 5th International Conference on Cryptology and Information Security in Latin America*, Habana, Cuba, September 20–22, 2017. Springer, Heidelberg, Germany.

[BHMS16] Colin Boyd, Britta Hale, Stig Frode Mjølsnes, and Douglas Stebila. From stateless to stateful: Generic authentication and authenticated encryption constructions with application to TLS. In Kazue Sako, editor, *Topics in Cryptology – CT-RSA 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 55–71, San Francisco, CA, USA, February 29 – March 4, 2016. Springer, Heidelberg, Germany.

[BJS16] Christina Brzuska, Håkon Jacobsen, and Douglas Stebila. Safely exporting keys from secure channels: On the security of EAP-TLS and TLS key exporters. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 670–698, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[BKN02] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In Vijayalakshmi Atluri, editor, *ACM CCS 02: 9th Conference on Computer and Communications Security*, pages 1–11, Washington D.C., USA, November 18–22, 2002. ACM Press.

[BKN04] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 7(2):206–241, 2004.

[BL16a] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16: 23rd Conference on Computer and Communications Security*, pages 456–467, Vienna, Austria, October 24–28, 2016. ACM Press.

[BL16b] Karthikeyan Bhargavan and Gaëtan Leurent. Transcript collision attacks: Breaking authentication in TLS, IKE and SSH. In *ISOC Network and Distributed System Security Symposium – NDSS 2016*, San Diego, CA, USA, February 21–24, 2016. The Internet Society.

[Bla16] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 1(1-2):1–135, 2016.

[BM97] Simon Blake-Wilson and Alfred Menezes. Entity authentication and authenticated key transport protocols employing asymmetric techniques. In Bruce Christianson, Bruno Crispo, T. Mark A. Lomas, and Michael Roe, editors, *Security Protocols, 5th International Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 137–158, Paris, France, April 7–9, 1997. Springer.

[BMM+15] Christian Badertscher, Christian Matt, Ueli Maurer, Phillip Rogaway, and Björn Tackmann. Augmented secure channels and the goal of the TLS 1.3 record layer. In Man Ho Au and Atsuko Miyaji, editors, *ProvSec 2015: 9th International Conference on Provable Security*, volume 9451 of *Lecture Notes in Computer Science*, pages 85–104, Kanazawa, Japan, November 24–26, 2015. Springer, Heidelberg, Germany.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany.

[Bon98] Dan Boneh. The decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *Lecture Notes in Computer Science*. Springer, Heidelberg, Germany, 1998. Invited paper.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.

[BPS15] Guy Barwell, Daniel Page, and Martijn Stam. Rogue decryption failures: Reconciling AE robustness notions. In Jens Groth, editor, *15th IMA International Conference on Cryptography and Coding*, volume 9496 of *Lecture Notes in Computer Science*, pages 94–111, Oxford, UK, December 15–17, 2015. Springer, Heidelberg, Germany.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany.

[BR95] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: The three party case. In Frank Thomson Leighton and Allan Borodin, editors, *27th Annual ACM Symposium on Theory of Computing*, pages 57–66, Las Vegas, NV, USA, May 29 – June 1, 1995. ACM Press.

[BR00] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

[Brz13] Christina Brzuska. *On the Foundations of Key Exchange*. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany, 2013. `http://tuprints.ulb.tu-darmstadt.de/3414/`.

[BSJ+17] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 619–650, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[BSWW13] Christina Brzuska, Nigel P. Smart, Bogdan Warinschi, and Gaven J. Watson. An analysis of the EMV channel establishment protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on*

Computer and Communications Security, pages 373–386, Berlin, Germany, November 4–8, 2013. ACM Press.

[BSY17] Hanno Böck, Juraj Somorovsky, and Craig Young. Return of bleichenbacher's oracle threat. `https://robotattack.org`, December 2017.

[BWJM97] Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In Michael Darnell, editor, *6th IMA International Conference on Cryptography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pages 30–45, Cirencester, UK, December 17–19, 1997. Springer, Heidelberg, Germany.

[BWM99a] Simon Blake-Wilson and Alfred Menezes. Authenticated Diffie-Hellman key agreement protocols (invited talk). In Stafford E. Tavares and Henk Meijer, editors, *SAC 1998: 5th Annual International Workshop on Selected Areas in Cryptography*, volume 1556 of *Lecture Notes in Computer Science*, pages 339–361, Kingston, Ontario, Canada, August 17–18, 1999. Springer, Heidelberg, Germany.

[BWM99b] Simon Blake-Wilson and Alfred Menezes. Unknown key-share attacks on the station-to-station (STS) protocol. In Hideki Imai and Yuliang Zheng, editors, *PKC'99: 2nd International Workshop on Theory and Practice in Public Key Cryptography*, volume 1560 of *Lecture Notes in Computer Science*, pages 154–170, Kamakura, Japan, March 1–3, 1999. Springer, Heidelberg, Germany.

[BY03] Mihir Bellare and Bennet S. Yee. Forward-security in private-key cryptography. In Marc Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, April 13–17, 2003. Springer, Heidelberg, Germany.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/2000/067`.

[CF12] Cas J. F. Cremers and Michele Feltz. Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS 2012: 17th European Symposium on Research in Computer Security*, volume 7459 of *Lecture Notes in Computer Science*, pages 734–751, Pisa, Italy, September 10–12, 2012. Springer, Heidelberg, Germany.

[CGCD+17] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the Signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017*, pages 451–466, Paris, France, April 26–28, 2017. IEEE.

[CGCG16] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. On post-compromise security. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016*, pages 164–178, Lisbon, Portugal, June 27 – July 1, 2016. IEEE Computer Society.

[CGCG+17] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. Cryptology ePrint Archive, Report 2017/666, 2017. `http://eprint.iacr.org/2017/666`.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In Jeffrey Scott Vitter, editor, *30th Annual ACM Symposium on Theory of Computing*, pages 209–218, Dallas, TX, USA, May 23–26, 1998. ACM Press.

[Che11] Lily Chen. *Recommendation for Key Derivation through Extraction-then-Expansion*. National Institute of Standards and Technology, November 2011.

[CHH+17] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17: 24th Conference on Computer and Communications Security*, pages 1773–1788, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[CHSvdM16] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *2016 IEEE Symposium on Security and Privacy*, pages 470–485, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.

[CK02a] Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 143–161, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany. `http://eprint.iacr.org/2002/120/`.

[CK02b] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 565–582, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[Cod14] Codenomicon. The Heartbleed bug. `http://heartbleed.com`, April 2014.

[DA99] Tim Dierks and Christopher Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176, 7465, 7507, 7919.

[Deg16] Jean Paul Degabriele. Personal communication, May 2016.

[DF11] Özgür Dagdelen and Marc Fischlin. Security analysis of the extended access control protocol for machine readable travel documents. In Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic, editors, *ISC 2010: 13th International Conference on Information Security*, volume 6531 of *Lecture Notes in Computer*

Science, pages 54–68, Boca Raton, FL, USA, October 25–28, 2011. Springer, Heidelberg, Germany.

[DFGS15a] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15: 22nd Conference on Computer and Communications Security*, pages 1197–1210, Denver, CO, USA, October 12–16, 2015. ACM Press.

[DFGS15b] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. Cryptology ePrint Archive, Report 2015/914, 2015. `http://eprint.iacr.org/2015/914`.

[DFGS16] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 draft-10 full and pre-shared key handshake protocol. Cryptology ePrint Archive, Report 2016/081, 2016. `http://eprint.iacr.org/2016/081`.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DKXY02] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 65–82, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

[DKXY03] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Strong key-insulated signature schemes. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 130–144, Miami, FL, USA, January 6–8, 2003. Springer, Heidelberg, Germany.

[DLFK+17] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella Béguelin, Karthikeyan Bhargavan, Jianyang Pan, and Jean Karim Zinzindohoue. Implementing and proving the TLS 1.3 record layer. In *2017 IEEE Symposium on Security and Privacy*, pages 463–482, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press.

[DLXY12] Yevgeniy Dodis, Weiliang Luo, Shouhuai Xu, and Moti Yung. Key-insulated symmetric key cryptography and mitigating attacks against cryptographic cloud software. In Heung Youl Youm and Yoojae Won, editors, *ASIACCS 12: 7th ACM Symposium on Information, Computer and Communications Security*, pages 57–58, Seoul, Korea, May 2–4, 2012. ACM Press.

[DP10] Jean Paul Degabriele and Kenneth G. Paterson. On the (in)security of IPsec in MAC-then-encrypt configurations. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10: 17th Conference on Computer and Communications Security*, pages 493–504, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.

[DR06] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176, 7465, 7507, 7919.

[DR08] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919.

[DS15] Benjamin Dowling and Douglas Stebila. Modelling ciphersuite and version negotiation in the TLS protocol. In Ernest Foo and Douglas Stebila, editors, *ACISP 15: 20th Australasian Conference on Information Security and Privacy*, volume 9144 of *Lecture Notes in Computer Science*, pages 270–288, Wollongong, NSW, Australia, June 29 – July 1, 2015. Springer, Heidelberg, Germany.

[Duo11] Thai Duong. BEAST. `http://vnhacker.blogspot.com.au/2011/09/beast.html`, September 2011.

[DVOW92] Whitfield Diffie, Paul C. Van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[Dwo07] Morris Dworkin. SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication, National Institute of Standards & Technology, November 2007.

[DY83] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Trans. Information Theory*, 29(2):198–207, 1983.

[EMV12] EMVCo LLC. EMV ECC key establishment protocols. `http://legacy.emvco.com/specifications.aspx?id=243`, 2012.

[FG14] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14: 21st Conference on Computer and Communications Security*, pages 1193–1204, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

[FG17] Marc Fischlin and Felix Günther. Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017*, pages 60–75, Paris, France, April 26–28, 2017. IEEE.

[FGM+97] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2068 (Proposed Standard), January 1997. Obsoleted by RFC 2616.

[FGMP15] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data is a stream: Security of stream-based channels. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 545–564, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[FGMP17] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data Is a Stream: Security of Stream-Based Channels. Cryptology ePrint Archive, Report 2017/1191, 2017. `https://eprint.iacr.org/2017/1191`.

[FGSW16] Marc Fischlin, Felix Günther, Benedikt Schmidt, and Bogdan Warinschi. Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In *2016 IEEE Symposium on Security and Privacy*, pages 452–469, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press.

[FKK11] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic), August 2011.

[Fou15] Cédric Fournet. [IETF TLS mailing list] resetting the sequence number to zero for each record key. (#379). `https://mailarchive.ietf.org/arch/msg/tls/extoO9ETJLnEm3MRDTO23x7ODFM`, December 2015.

[FR14] Roy T. Fielding and Julian F. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230 (Proposed Standard), June 2014.

[FW09] Pooya Farshim and Bogdan Warinschi. Certified encryption revisited. In Bart Preneel, editor, *AFRICACRYPT 09: 2nd International Conference on Cryptology in Africa*, volume 5580 of *Lecture Notes in Computer Science*, pages 179–197, Gammarth, Tunisia, June 21–25, 2009. Springer, Heidelberg, Germany.

[GHJL17] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT key exchange with full forward secrecy. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 519–548, Paris, France, May 8–12, 2017. Springer, Heidelberg, Germany.

[GKS13] Florian Giesen, Florian Kohlar, and Douglas Stebila. On the security of TLS renegotiation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 387–398, Berlin, Germany, November 4–8, 2013. ACM Press.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GM17] Felix Günther and Sogol Mazaheri. A formal treatment of multi-key channels. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 587–618, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[GMP+08] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. Universally composable security analysis of TLS. In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *ProvSec 2008: 2nd International Conference on Provable Security*, volume 5324 of *Lecture Notes in Computer Science*, pages 313–327, Shanghai, China, October 31 – November 1, 2008. Springer, Heidelberg, Germany.

[GMSS08] Sebastian Gajek, Mark Manulis, Ahmad-Reza Sadeghi, and Jörg Schwenk. Provably secure browser-based user-aware mutual authentication over TLS. In Masayuki Abe and Virgil Gligor, editors, *ASIACCS 08: 3rd ACM Symposium on Information, Computer and Communications Security*, pages 300–311, Tokyo, Japan, March 18–20, 2008. ACM Press.

[Gün90] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 29–37, Houthalen, Belgium, April 10–13, 1990. Springer, Heidelberg, Germany.

[Gün15] Felix Günther. [IETF TLS mailing list] A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. `https://mailarchive.ietf.org/arch/msg/tls/cgc7cqEYvZJjw56WSr8_xP9gDIM`, September 2015.

[HC09] Katrin Hoeper and Lily Chen. SP 800-120. Recommendation for EAP Methods Used in Wireless Network Access Authentication. NIST Special Publication, National Institute of Standards & Technology, September 2009.

[HJLS17] Britta Hale, Tibor Jager, Sebastian Lauer, and Jörg Schwenk. Simple security definitions for and constructions of 0-RTT key exchange. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17: 15th International Conference on Applied Cryptography and Network Security*, volume 10355 of *Lecture Notes in Computer Science*, pages 20–38, Kanazawa, Japan, July 10–12, 2017. Springer, Heidelberg, Germany.

[Int15] International Civil Aviation Organization (ICAO). Machine Readable Travel Documents, Part 11, Security Mechanisms for MRTDs. Doc 9303, 2015. Seventh Edition.

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[JP02] Markus Jakobsson and David Pointcheval. Mutual authentication for low-power mobile devices. In Paul F. Syverson, editor, *FC 2001: 5th International Conference on Financial Cryptography*, volume 2339 of *Lecture Notes in Computer Science*, pages 178–195, Grand Cayman, British West Indies, February 19–22, 2002. Springer, Heidelberg, Germany.

[JSS15] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. On the security of TLS 1.3 and QUIC against weaknesses in PKCS#1 v1.5 encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15: 22nd Conference on Computer and Communications Security*, pages 1185–1196, Denver, CO, USA, October 12–16, 2015. ACM Press.

[Kah96] David Kahn. *The Code-Breakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, 1996.

[Kal98] Burt Kaliski. PKCS #1: RSA Encryption Version 1.5. RFC 2313 (Informational), March 1998. Obsoleted by RFC 2437.

[Kat10] Jonathan Katz. *Digital Signatures*. Springer, Heidelberg, Germany, 2010.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997. Updated by RFC 6151.

[KE10] Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (Informational), May 2010.

[KL08] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2008.

[KMO+14] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Bjoern Tackmann, and Daniele Venturi. (De-)constructing TLS. Cryptology ePrint Archive, Report 2014/020, 2014. http://eprint.iacr.org/2014/020.

[KP05] Caroline Kudla and Kenneth G. Paterson. Modular security proofs for key agreement protocols. In Bimal K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 549–565, Chennai, India, December 4–8, 2005. Springer, Heidelberg, Germany.

[KPB03] Tadayoshi Kohno, Adriana Palacio, and John Black. Building secure cryptographic transforms, or how to encrypt and MAC. Cryptology ePrint Archive, Report 2003/177, 2003. http://eprint.iacr.org/2003/177.

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[KR17] Ralf Küsters and Daniel Rausch. A framework for universally composable Diffie-Hellman key exchange. In *2017 IEEE Symposium on Security and Privacy*, pages 881–900, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press.

[Kra05] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[Kra16a] Hugo Krawczyk. [IETF TLS mailing list] Re: Call for consensus: Removing DHE-based 0-RTT. https://mailarchive.ietf.org/arch/msg/tls/xmnvrKEQkEbD-u8HTeQkyitmclY, March 2016.

[Kra16b] Hugo Krawczyk. A unilateral-to-mutual authentication compiler for key exchange (with applications to client authentication in TLS 1.3). In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16: 23rd Conference on Computer and Communications Security*, pages 1438–1450, Vienna, Austria, October 24–28, 2016. ACM Press.

[KS05] Stephen Kent and Karen Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005. Updated by RFCs 6040, 7619.

[KSS13] Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DH and TLS-RSA in the standard model. Cryptology ePrint Archive, Report 2013/367, 2013. http://eprint.iacr.org/2013/367.

[KT11] Ralf Küsters and Max Tuengerthal. Composition theorems without pre-established session identifiers. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 41–50, Chicago, Illinois, USA, October 17–21, 2011. ACM Press.

[KW15] Hugo Krawczyk and Hoeteck Wee. The OPTLS protocol and TLS 1.3. Cryptology ePrint Archive, Report 2015/978, 2015. http://eprint.iacr.org/2015/978.

[KW16] Hugo Krawczyk and Hoeteck Wee. The OPTLS protocol and TLS 1.3. In *2016 IEEE European Symposium on Security and Privacy, EuroS&P 2016*, pages 81–96, Saarbrücken, Germany, March 21–24, 2016. IEEE.

[Lad14] Watson Ladd. [IETF TLS mailing list] Kill Finished (and other tricks for hardware). https://www.ietf.org/mail-archive/web/tls/current/msg12162.html, April 2014.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[Lan14] Adam Langley. Comment at the Real World Crypto (RWC) Workshop 2014, New York, NY, USA, January 13–15, 2014.

[LC13] Adam Langley and Wan-Teh Chang. QUIC Crypto, June 2013. Revision 20130620.

[LC16] Adam Langley and Wan-Teh Chang. QUIC Crypto. https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/, December 2016. Revision 20161206.

[LJBN15] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. How secure and quick is QUIC? Provable security and performance analyses. In *2015 IEEE Symposium on Security and Privacy*, pages 214–231, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007: 1st International Conference on Provable Security*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16, Wollongong, Australia, November 1–2, 2007. Springer, Heidelberg, Germany.

[LM06] Kristin Lauter and Anton Mityagin. Security analysis of KEA authenticated key exchange protocol. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 378–394, New York, NY, USA, April 24–26, 2006. Springer, Heidelberg, Germany.

[LP16] Atul Luykx and Kenneth G. Paterson. Limits on authenticated encryption use in TLS. http://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf, 2016.

[LRW+17] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The QUIC transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017*, pages 183–196, Los Angeles, CA, USA, August 21–25, 2017. ACM.

[LSY+14] Yong Li, Sven Schäge, Zheng Yang, Florian Kohlar, and Jörg Schwenk. On the security of the pre-shared key ciphersuites of TLS. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 669–684, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany.

[LXZ+16] Xinyu Li, Jing Xu, Zhenfeng Zhang, Dengguo Feng, and Honggang Hu. Multiple handshakes security of TLS 1.3 candidates. In *2016 IEEE Symposium on Security and Privacy*, pages 486–505, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press.

[Mac17] Colm MacCárthaigh. [IETF TLS mailing list] Security review of TLS1.3 0-RTT. `https://mailarchive.ietf.org/arch/msg/tls/mHxi-O3du9OQHkc6CBWBpc_KBpA`, May 2017.

[Mat16] Shin'ichiro Matsuo. [IETF TLS mailing list] Formal Verification of TLS 1.3 Full Handshake Protocol Using ProVerif (Draft-11). `https://mailarchive.ietf.org/arch/msg/tls/NXGYUUXCD2b9WwBRWbvrccjjdyI`, February 2016.

[MDK14] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This POODLE bites: Exploiting the SSL 3.0 fallback. `https://www.openssl.org/~bodo/ssl-poodle.pdf`, September 2014.

[miT] miTLS: A Verified Reference Implementation of TLS. `http://mitls.org/`.

[MP17a] Giorgia Azzurra Marson and Bertram Poettering. Security notions for bidirectional channels. *IACR Transactions on Symmetric Cryptology*, 2017(1):405–426, 2017.

[MP17b] Giorgia Azzurra Marson and Bertram Poettering. With one it is easy, with many it gets complicated: Understanding channel security for groups. Cryptology ePrint Archive, Report 2017/786, 2017. `http://eprint.iacr.org/2017/786`.

[MR11] Ueli Maurer and Renato Renner. Abstract cryptography. In Bernard Chazelle, editor, *ICS 2011: 2nd Innovations in Computer Science*, pages 1–21, Tsinghua University, Beijing, China, January 7–9, 2011. Tsinghua University Press.

[MSW08] Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. A modular security analysis of the TLS handshake protocol. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 55–73, Melbourne, Australia, December 7–11, 2008. Springer, Heidelberg, Germany.

[MT10] Ueli Maurer and Björn Tackmann. On the soundness of authenticate-then-encrypt: formalizing the malleability of symmetric encryption. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10: 17th Conference on Computer and Communications Security*, pages 505–515, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, 1996.

[Nam02] Chanathip Namprempre. Secure channels based on authenticated encryption schemes: A simple characterization. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer*

Science, pages 515–532, Queenstown, New Zealand, December 1–5, 2002. Springer, Heidelberg, Germany.

[NRS14] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 257–274, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In Harriet Ortiz, editor, *22nd Annual ACM Symposium on Theory of Computing*, pages 427–437, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[OP01] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118, Cheju Island, South Korea, February 13–15, 2001. Springer, Heidelberg, Germany.

[Ope] The OpenSSL Project. `https://www.openssl.org`.

[Orm17] Tavis Ormandy. cloudflare: Cloudflare reverse proxies are dumping uninitialized memory. `https://bugs.chromium.org/p/project-zero/issues/detail?id=1139`, February 2017.

[Poe16] Bertram Poettering. Personal communication, July 2016.

[Pos80] Jon Postel. User Datagram Protocol. RFC 768 (Internet Standard), August 1980.

[Pos81a] Jon Postel. Internet Protocol. RFC 791 (Internet Standard), September 1981. Updated by RFCs 1349, 2474, 6864.

[Pos81b] Jon Postel. Transmission Control Protocol. RFC 793 (Internet Standard), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[Pro] Proverif: Cryptographic protocol verifier in the formal model. `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`.

[PRS11] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

[PvdM16] Kenneth G. Paterson and Thyla van der Merwe. Reactive and proactive standardisation of TLS. In Lidong Chen, David A. McGrew, and Chris J. Mitchell, editors, *Security Standardisation Research: Third International Conference (SSR 2016)*, volume 10074 of *Lecture Notes in Computer Science*, pages 160–186, Gaithersburg, MD, USA, December 5–6, 2016. Springer.

[PW10] Kenneth G. Paterson and Gaven J. Watson. Plaintext-dependent decryption: A formal security treatment of SSH-CTR. In Henri Gilbert, editor, *Advances in*

Cryptology – EUROCRYPT 2010, volume 6110 of *Lecture Notes in Computer Science*, pages 345–361, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[PZS+13] W. Michael Petullo, Xu Zhang, Jon A. Solworth, Daniel J. Bernstein, and Tanja Lange. MinimaLT: minimal-latency networking through better security. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 425–438, Berlin, Germany, November 4–8, 2013. ACM Press.

[QUI] QUIC, a multiplexed stream transport over UDP. `https://www.chromium.org/quic`.

[QWG] IETF QUIC working group. `https://datatracker.ietf.org/wg/quic/about/`.

[Res15a] Eric Rescorla. [IETF TLS mailing list] 0-RTT and Anti-Replay. `https://www.ietf.org/mail-archive/web/tls/current/msg15594.html`, March 2015.

[Res15b] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-05. `https://tools.ietf.org/html/draft-ietf-tls-tls13-05`, March 2015.

[Res15c] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-07. `https://tools.ietf.org/html/draft-ietf-tls-tls13-07`, July 2015.

[Res15d] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-08. `https://tools.ietf.org/html/draft-ietf-tls-tls13-08`, August 2015.

[Res15e] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-10. `https://tools.ietf.org/html/draft-ietf-tls-tls13-10`, October 2015.

[Res15f] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-11. `https://tools.ietf.org/html/draft-ietf-tls-tls13-11`, December 2015.

[Res15g] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-dh-based. `https://github.com/ekr/tls13-spec/blob/ietf92_materials/draft-ietf-tls-tls13-dh-based.txt`, March 2015.

[Res16a] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-12. `https://tools.ietf.org/html/draft-ietf-tls-tls13-12`, March 2016.

[Res16b] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-13. `https://tools.ietf.org/html/draft-ietf-tls-tls13-13`, May 2016.

[Res16c] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-14. `https://tools.ietf.org/html/draft-ietf-tls-tls13-14`, July 2016.

[Res16d] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-18. https://tools.ietf.org/html/draft-ietf-tls-tls13-18, October 2016.

[Res16e] Eric Rescorla. TLS 1.3 — draft-ietf-tls-tls13-12 (presentation at ietf 95 meeting). https://www.ietf.org/proceedings/95/slides/slides-95-tls-2.pdf, April 2016.

[Res18] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-24. https://tools.ietf.org/html/draft-ietf-tls-tls13-24, February 2018.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 02: 9th Conference on Computer and Communications Security*, pages 98–107, Washington D.C., USA, November 18–22, 2002. ACM Press.

[Rog06] Phillip Rogaway. Formalizing human ignorance. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06: 1st International Conference on Cryptology in Vietnam*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228, Hanoi, Vietnam, September 25–28, 2006. Springer, Heidelberg, Germany.

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

[Sho99] Victor Shoup. On formal models for secure key exchange. Cryptology ePrint Archive, Report 1999/012, 1999. http://eprint.iacr.org/1999/012.

[Sho06] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs, 2006. Manuscript.

[Sig] Signal protocol: Technical documentation. https://whispersystems.org/docs/.

[SMCB12] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *IEEE 25th Computer Security Foundations Symposium, CSF 2012*, pages 78–94, Cambridge, MA, USA, June 25–27, 2012. IEEE Computer Society.

[SP13] Ben Smyth and Alfredo Pironti. Truncating TLS connections to violate beliefs in web applications. In *7th USENIX Workshop on Offensive Technologies, WOOT '13*, Washington, D.C., USA, August 13, 2013. USENIX Association. (first appeared at Black Hat USA 2013).

[SYHY16] Hideki Sakurada, Kazuki Yoneyama, Yoshikazu Hanatani, and Maki Yoshida. Analyzing and fixing the QACCE security of QUIC. In Lidong Chen, David A. McGrew, and Chris J. Mitchell, editors, *Security Standardisation Research: Third International Conference (SSR 2016)*, volume 10074 of *Lecture Notes in Computer Science*, pages 1–31, Gaithersburg, MD, USA, December 5–6, 2016. Springer.

[TLS17] TLS:DIV (TLS 1.3: Design, Implementation & Verification) Workshop at IEEE EuroS&P / EUROCRYPT 2017. https://www.mitls.org/tls:div/, April 2017.

[TRO16a] TRON (TLS1.3 – Ready or Not?) Workshop at NDSS 2016. https://www.ndss-symposium.org/ndss2016/tron-workshop-programme/, February 2016.

[TRO16b] IEEE Security & Privacy 2016 TLS 1.3 Meetup. https://www.mitls.org/tron2/, May 2016.

[TT17] Martin Thomson and Sean Turner. Using Transport Layer Security (TLS) to Secure QUIC – draft-ietf-quic-tls-06. https://tools.ietf.org/html/draft-ietf-quic-tls-06, September 2017.

[WTSB16] David J. Wu, Ankur Taly, Asim Shankar, and Dan Boneh. Privacy, discovery, and authentication for the internet of things. In Ioannis G. Askoxylakis, Sotiris Ioannidis, Sokratis K. Katsikas, and Catherine A. Meadows, editors, *ESORICS 2016: 21st European Symposium on Research in Computer Security, Part II*, volume 9879 of *Lecture Notes in Computer Science*, pages 301–319, Heraklion, Greece, September 26–30, 2016. Springer, Heidelberg, Germany.

[YL06a] Tatu Ylonen and Chris Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006.

[YL06b] Tatu Ylonen and Chris Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), January 2006. Updated by RFC 6668.